
University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

# Modeling methods with implicitly defined objects

State of the Art and Concept of Doctoral Thesis

Karel Uhlíř

Technical Report No. DCSE/TR-2003-04
February, 2003

Distribution: public

Technical Report No. DCSE/TR-2003-04
February 2003

# Modeling methods with implicitly defined objects

Karel Uhlíř

---

## Abstract

A new trend in the usage of functional modeling in computer science has been formed during several last years. One of the basic fields developing in a lot of departments is modeling with the implicitly defined objects. The implicit definition (in contrast to the parametric definition of the object) is the most compact description of the model, which exactly defines the object surface (volumetric data). One of the basic modeling methods is modeling with the Boolean operations. The Boolean operations allow creating the CSG tree structure from the implicitly defined objects. CSG tree can be further used in the visualization methods. Visualization of an implicitly defined object is possible by using several methods. One group of these methods represents direct methods like ray tracing or ray casting, in the second group belong surface approximation methods with triangle mesh like Marching cubes or Marching tetrahedra. Since, this kind of description has many benefits, there is a tendency to represent the objects described by triangle mesh with implicit equation. Area of the implicit modeling is very large.

This offered work contains an overview of methods used for implicitization of objects defined by the polygonal mesh or point-cloud data. The mathematical methods for implicitization are briefly introduced, too. The main goal of the research is oriented to the Radial Basis Function method. Basic aspects, possible solutions, advantages and disadvantages of presented algorithms are discussed. The outlook of our previous work and future work is presented.

---

This work was supported by identification of grant or project.

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Copyright © 2003 University of West Bohemia in Pilsen, Czech Republic

## Content

# Implicit surface

## 1 Introduction

Implicit surfaces are used in computer graphics science for very long time. The first complex shapes that were created using implicit functions appeared nearly 20 years ago. The meaning of implicit surfaces has significantly increased in recent years. The implicit surfaces can be used for object description in space of any dimension. In this paper we will operate only in 2 and 3 dimensional space. All objects can be defined by particular mathematical form. Implicit surfaces are becoming more and more popular in computer graphics and object or models expressed by the implicit equation are good opponents to objects defined by parametric equations.

## 1.2 Object description

A surface *O* of an object can be represented implicitly by a set of points, which satisfy

$$O = \{p \in \Re^3 : f(p) = c\} \tag{1}$$

The roots of the equation $f(p)-c=0$ determine a set of points and represent the surface of the implicit object. It means that *implicit surface* is the set of the solutions of an equation $f(p) = 0$. The function given a point $p$, returns scalar value $c$ that specifies whether the point is inside, on boundary, or outside the shape that is being described. We will use positive function values, $f(p) > 0$, to mean that the point $p$ is inside a shape, and $f(p) < 0$ will mean that the point is outside. Note that there is used notation $p = (x, y, z)$ for points $p \in \Re^3$.

An implicit volume is the set of the solutions of an inequality of the form $f(p) \geq 0$ and the ambient space in the form $f(p) \leq 0$. It is possible to introduce an inverse definition of implicit volume and the ambient space, which is sometimes used in other literature. The inverse definition just changes the signs in the implicit equation defining object *O*.

We can take the sphere object as an example of the object definition. The sphere may be described in both parametric and implicit form. The parametric form is

$$f(\alpha, \beta) = (\cos\alpha \cos\beta, \sin\alpha, \cos\alpha \sin\beta), \ \alpha \in [0, \pi], \ \beta \in [0, 2\pi] \tag{2}$$

and the implicit form is

$$f(x, y, z) = x^2 + y^2 + z^2 - 1 \tag{3}$$

It is obvious, that the implicit representation of sphere is more compact, than equal parametric form. It can be seen from equation (3), that there are positive values outside object and negative values inside the unit sphere. There is used inverse definition of

implicit surface in the following text, therefore we are going to introduce the inverse definition to equation (3).

$$f(x,y,z) = x^2 - y^2 - z^2 + 1. \quad \textbf{(4)}$$

Both parametric and implicit surfaces may represent complex objects. For implicit surfaces, complexity can be specified by an arbitrarily complex black box function or by an algebraic function with an arbitrary number of terms. Because implicit surfaces conveniently define volume, they are used frequently in CSG-based solid modelers.

## 1.3 Modeling

### 1.3.1 CSG

Constructive solid geometry represents an important class of implicit models. CSG modeling is a hierarchical modeling where all objects are defined in terms of other objects. CSG objects are built from point sets that are defined by primitive functions (primitives) and combined by Boolean operators. The primitives can be polygons, simple geometric objects, such as the sphere or more complicated elements, such as parametric patches or blended objects. Operators and the primitives make the hierarchical structure called CSG tree.

**Figure 1: CSG primitives**.

The basic primitives are nodes in a CSG tree. The primitive in a node can be simple geometric object, such as sphere, torus, cone [Figure 1] or very complex object but finally a CSG tree is a single implicit model [Figure 2].

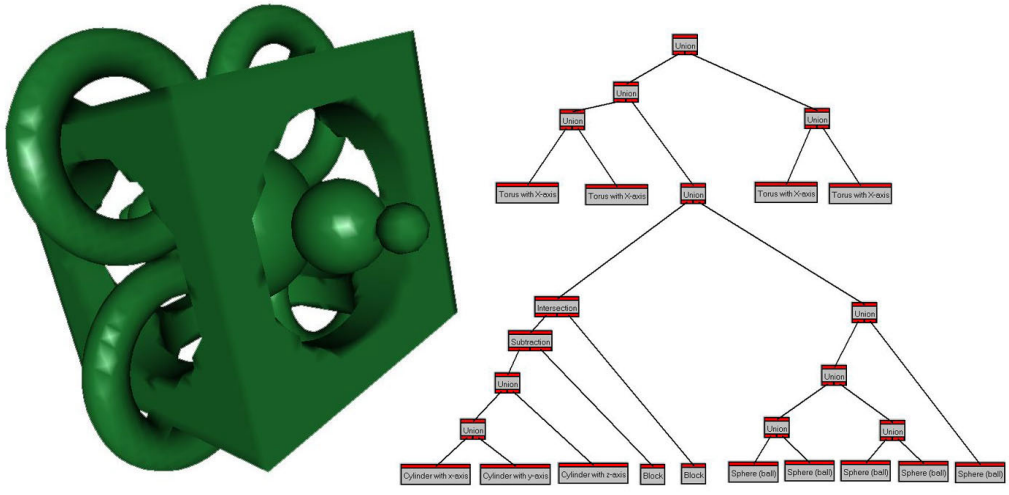



**Figure 2: Complex model and its CSG tree.**

More complicated parts of a CSG tree are operations which provide connection of all nodes (primitives) and representation of their modifications. The basic operations are Boolean operations as union, intersection, subtraction or negation. The definitions of those operations can differ and depend on the desired degree of continuity. The simplest forms of Boolean operations are

| | | |
|---|---|---|
| Union | $f_1 \cup f_2 = \max(f_1, f_2)$, | **(5)** |
| Intersection | $f_1 \cap f_2 = \min(f_1, f_2)$, | **(6)** |
| Subtraction | $f_1 - f_2 = \min(f_1, -f_2)$. | **(7)** |

Equations (5)(6) and (7) are very convenient for calculations but are not $C^1$ continuous for f1 = $f_2$ [Pasko95]. Figure 3 shows an example of the union, intersection and subtraction operator applied to two spheres [Dekkers97].

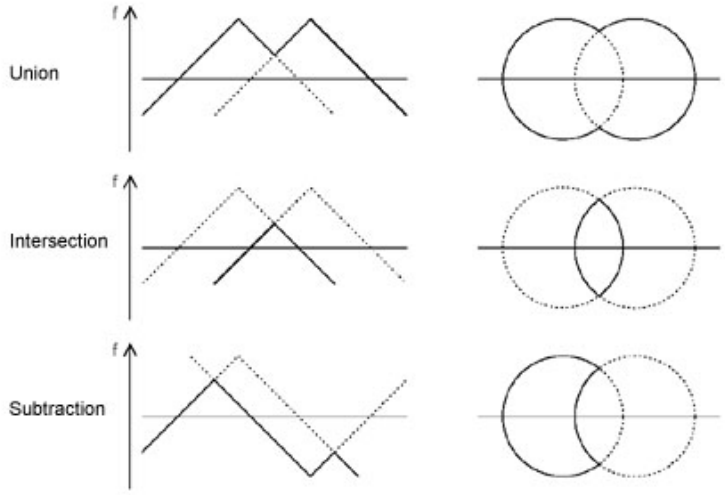



**Figure 3: Set operations. [Dekkers97]**

We have to use the the following definition of Boolean operations, if $C^m$ continuity must be achieved [Pasko95]:

| | | |
|---|---|---|
| Union | $f_1 \cup f_2 = (f_1 + f_2 + \sqrt{f_1^2 + f_2^2})(f_1^2 + f_2^2)^{\frac{m}{2}}$, | **(8)** |
| Intersection | $f_1 \cup f_2 = (f_1 + f_2 - \sqrt{f_1^2 + f_2^2})(f_1^2 + f_2^2)^{\frac{m}{2}}$. | **(9)** |

The basic Boolean operations can be extended to the definition mentioned above (Eq. (5),(6),(7) or (8),(9)) can be added transition function (10)[Pasko95]. The transition function is called the *Blending function* [Figure 4]. The Blending function definition can use the analogy of spatial temperature distribution: if one moves away from a heat source, the temperature drops [Wyvill86]. Different definition of the Blending function can be found in [Pasko95] or [Dekkers97].

Blending Union

$$f_1 \cup f_2 = \max(f_1, f_2) + d(f_1, f_2)$$
$$d(f_1, f_2) = \frac{a_0}{1 + \left(\frac{f_1}{a_1}\right)^2 + \left(\frac{f_2}{a_2}\right)^2} \quad \textbf{(10)}$$

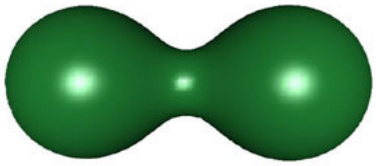


**Figure 4: Blending function at the two spheres. [Dekkers97]**

A transformation can be used like an operation in CSG tree, too. Position, form and parameters of the primitive can be modified with rotation, scaling, shifting, twisting etc.

The CSG tree is a structure useful in a lot of modelers and CAD systems. This structure gives us information about hierarchy of a model and can be used in visualization methods, skeleton (convolution) modeling or any manipulation with surface (collision detection). More information about some modelers can be found in [Adzhiev99], [Adzhiev00], [HyperFun99], [Pasko93], [Uhlir03], [Uhlir02] or [Uhlir01].

### 1.3.2 Skeleton

Skeleton, a standard CAD representation, has become a popular construct for implicit design. A typical skeleton is *hierarchical*. Each skeletal element, or *limb*, may support one or more descendent limbs. The limb not descendent from any other is the *root* of the skeleton. The connection point between limbs is a *joint*. A skeleton is often represented as a direct, acyclic graph. A skeleton may be constructed interactively or digitized from a physical object. It may be manipulated by changing joint transformation.

The primitive is defined as those points that are at a particular distance from the skeletal element (for example, the skeleton of a sphere is its center). Finally there exist two ways to define an implicit surface from a skeleton.

First is a *distance surface* [Bloomenthal97]. A distance surface is a surface that is defined by distance to some set of base surfaces (or skeletal elements) such as points, line segments, polygons, or any curve, surface or volume. Its means that the field value at a given point P is calculated from the distance between P and the closest point on the skeleton. For a curve is a distance a generalized cylinder in three-space. Blending the contributions of several skeleton elements is then usually performed by summing their field contributions.

Second way is a *convolution surface*. In this representation, the field value at a point P is calculated by integrating all the contributions from the different points on the skeleton. Smooth complex surface can be created by summing the integrals if individual field contributions of relatively simple skeletal elements [Bloomenthal91], [Shersyuk99].

More information about skeletal modeling in a field of implicit surfaces can be found in [Angelidis02], [Bloomenthal97], [Bloomenthal91] or [Rigaudiere99].

## 1.4 Visualization

Visualizing implicit surfaces typically consists of finding the zero-set of *f*, which may be performed either by polygonizing the surface or by direct ray tracing. A lot of techniques exist for the visualization and the rendering of the implicitly defined surfaces. These techniques can be divided into two categories: direct, indirect.

### 1.4.1 Direct

These methods make the direct visualization of implicitly defined object. Rendering directly from the implicit model reduce the volume data, and it is possible to zoom in on fine detail in a model without losing quality. If we are talking about direct method we assume ray tracing. Although slower, ray tracing provides a direct, accurate, and elegant method for investigating a much larger variety of implicit surfaces. In ray tracing processing must be find the intersection even if the original model is not defined implicitly. If we start with an implicit model, we already have this equation in principle. Implicit model can be CSG tree too. CSG tree is a single implicit model and, as such, it can be ray traced directly. In principle, we must find the intersection of ray with every primitive in the CSG model. For making pictures, we need only the first intersection with the CSG tree. The complete classification of the CSG tree is not needed. The ray tracing method optimized to find the first valid intersection quickly could be found in [Wyvill87]. Another ray tracing method can be the sphere tracing method [Hart96]. This technique for rendering implicit surface uses geometric distance and the function must be continuous and Lipschitz. Scan-line rendering technique [Hubb98] works with Lipschitz condition, too. This technique is also viable for fast prototyping of implicit surface.

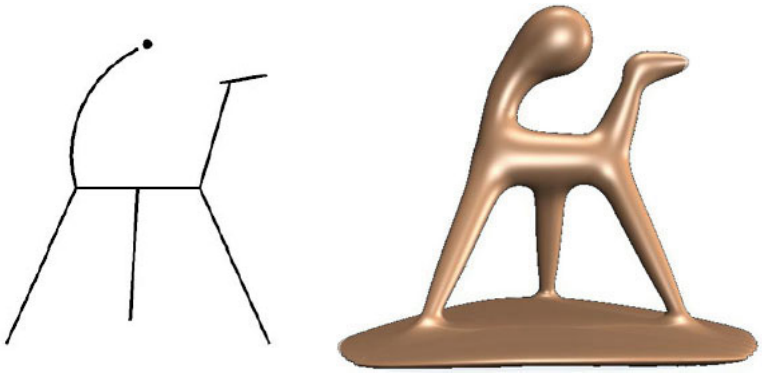


**Figure 5: An example of modeling with convolution surfaces. The left image is a skeleton and the right image is a ray-traced convolution surface [Shersyuk99]**

### 1.4.2 Indirect

Indirect methods polygonize the implicit surface to a given tolerance, allowing the use of existing polygon-rendering techniques and hardware for interactive inspection. Polygonization of an iso surface of a function of three variables (or implicit surface) includes sampling the function at the selected points, estimating the position of the mesh vertices, and connecting them to the polygons. Although polygonization transforms implicit surfaces into a representation easily rendered and incorporated into graphics systems, polygonizations are typically not guaranteed and may not accurately detect disconnected or detailed sections of the implicit surface. Production-rendering systems tend to polygonize surfaces, resulting in large time and memory overheads to represent accurately an otherwise simple implicit model. For many implicitly defined surfaces,

polygonization followed by polygon rendering is more efficient than direct rendering methods.

Jules Bloomenthal [Bloomenthal97] introduced the basic method for the surface polygonization. His method is “walking” on the implicit surface and evaluating the implicit function in node of the regular grid which divides space of evaluation. The visualization can be made with e.g. marching cubes or marching tetrahedra together with OpenGL. Interesting method for creating set of triangles from isosurface is marching triangles [Hartmann98, Cermak02a, Cermak02b], too. Some of these methods are used without any modification. It means improvements in accurate polygonization of implicit surface with sharp features [Ohtake01], adaptive sampling [Velho96] where highly curved parts are detected and then these cells are subdivided [Bloomenthal88] or optimization of the methods in the performance and the storage [Wyvill86].

Some of the visualization methods are based on application of a deformation on the basic surface (sphere etc.) to transform it into the required surface [Overveld93].

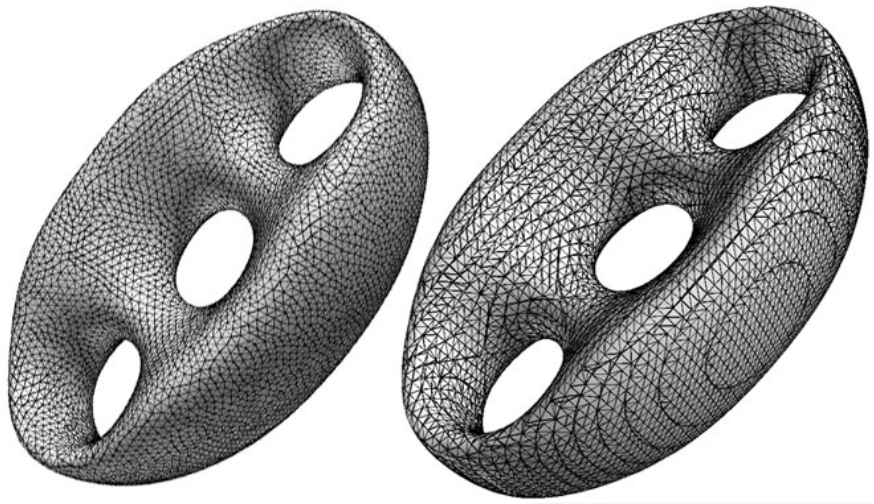


**Figure 6: Genus polygonized by the marching triangles method (left) and marching cubes method (right). (Marching triangles [Hartmann98], Marching cubes [Bloomenthal97]) [Cermak02a]**

# Implicitization

There will be discussed methods for creating implicit representation of arbitrary objects in this chapter. There exist two ways for a creation of implicit representation of object.

First way uses parametric expression of a primitive or a patch on the beginning as an input. The implicit representation of the object is generated by using symbolic operations for the parametric expression of object. The group of these methods is called *Variables elimination methods*.

The second way starts with polygonal mesh or point-cloud data. Iso-value, which provides information about particular point position, can be calculated from this representation directly. Then iso-value means, whether the point is inside, outside or on the surface. Below are analyzed basic properties of symbolic methods and will be elaborated method for generating implicit representation from polygonal mesh or point-cloud data.

## 2.1 Variables elimination methods

Elimination is a mathematical discipline for removing variables from system of equations. The results of these work has become very popular in the last 15 years. In [Hoffmann93] is made classification of the resultant method, the Gröbner basis method, and the Wu-Ritt method at the most well-know and major competing approach.

### 2.1.1 Gröbner basis method

This method is based on finding a Gröbner basis for an ideal I. The ideal I is an ordered set of polynomials (polynomial ideal), which meets a requirement of existence a Gröbner basis. Seeking reduced Gröbner basis bear on seeking exact solution of polynomial equations system. If polynomial equations system has a solution then the variables of system are eliminated and the original set of equations is transformed. The transformed set of equations can be easily solved. Seeking of the Gröbner basis for ideal I can be done with Buchberger's algorithm. This algorithm has a lot of modifications, because searching of the Gröbner basis is very computational expensive.

The transformation of the parametric expression of affine variety to the implicit can be successfully solved by using the Gröbner basis of an ideal. There exist two ways for solving transformation. These ways are based on a form of the variety entry. The variety can be entering either *polynomial parameterization* or *rational parameterization*.

**Ideal**

Set $I \subset k[x_1,\ldots,x_n]$ is called an ideal in $k[x_1,\ldots,x_n]$ if the following two conditions are true:

1. for all polynomials $f, g \in I$, it is necessary that $f+g \in I$ and
2. for all polynomials $f \in I$, it is necessary that $fg \in I$ for any $g \in k[x_1,\ldots,x_n]$.

Let $f_1,\ldots,f_s \in k[x_1,\ldots,x_n]$. Consider an ideal $I$ that contains all of $f_1,\ldots,f_s$. The set $I = \left\{\sum_{i=1}^{s} g_i f_i \mid g_i \in k[x_i,\ldots,x_n]\right\}$ is an ideal in $k[x_i,\ldots,x_n]$ and it is the smallest ideal in $k[x_i,\ldots,x_n]$ containing the set $\{f_1,\ldots,f_s\}$. This set is called a generating set or a basis for the ideal *I*.

**Ordering of the polynomials**

For the computation of the Gröbner basis the ordering of the terms in a polynomial is essential. Of interest is a total ordering on terms which is denoted by $\prec$ and which has following properties:

1. The ordering is compatible with a multiplication. For example, given tree terms $t, t_1$ and $t_2$, if $t_1 \prec t_2$ then $tt_1 \prec tt_2$.
2. For finite polynomials there can be no strictly decreasing infinite sequence of terms such as $t_1 \succ t_2 \succ \ldots$.

The following two ordering schemas are the most common ones.

*Lexicographic ordering*

It is ordering of the terms in a dictionary; its symbol is $\prec_l$. For example, given two terms $t_1$ and $t_2$ which are made up with two variables $x_1$ and $x_2$ where $x_1 \prec_l x_2$ the following lexicographical ordering results:

$$1 \prec_l x_1 \prec_l x_1^2 \prec_l x_1^3 \prec_l \ldots \prec_l x_2 \prec_l x_1 x_2 \prec_l x_1^2 x_2 \prec_l \ldots \prec_l x_2^2 \prec_l x_1 x_2^2 \prec_l x_1^2 x_2^2 \prec_l \ldots \quad \textbf{(11)}$$

Sometimes a reverse lexicographical ordering is used, too.

*Degree ordering*

This method first order the terms by their degrees and equal degree terms are then ordered lexicographically. If the same example like in case of lexicographical ordering is used, then ordering result is:

$$1 \prec_d x_1 \prec_d x_2 \prec_d x_1^2 \prec_d x_1 x_2 \prec_d x_2^2 \prec_d x_2 \prec_d x_1^3 \prec_d x_1^2 x_2 \prec_d x_1 x_2^2 \prec_d x_2^3 \prec_d \ldots \quad \textbf{(12)}$$

**Reduction of the polynomials**

For the calculation of the Gröbner basis it is important to perform a polynomial reduction. Before the polynomial reduction can be performed, an ordering $\prec$ of the terms has to be chosen. With the ordering $\prec$ the following components of a polynomial are defined:

*Leading monomial of a polynomial*

For every polynomial $f(x_1, x_2, \ldots, x_n)$ the leading monomial is given by the largest term in $f$ under $\prec$ which has non-zero coefficients. This monomial is denoted by *LM*(*f*).

*Leading coefficient of a polynomial*

The coefficient of the leading monomial [1]is then the leading coefficient which is denoted by *LC*(*f*).

[1] Often this term is called the head term of the polynomial.

*Leading term of a polynomial*

The leading term of a polynomial is given by the multiplication of leading monomial and leading coefficient and is denoted by *LT*(*f*).

$$LT(f) = LC(f)LM(f) \tag{13}$$

*Tail of a polynomial*

The tail term of a polynomial $f(x_1, x_2, \ldots, x_n)$ which is denoted by *TT*(*f*) is given by splitting the leading term from the polynomial *f*.

With the definitions above a polynomial $f(x_1, x_2, \ldots, x_n)$ can be rewritten in the following manner:

$$f = LT(f) + TT(f) = LC(f)LM(f) + TT(f) \tag{14}$$

**Polynomial reduction**

Given two polynomials $f(x_1, x_2, \ldots, x_n)$ and $g(x_1, x_2, \ldots, x_m)$, *g* reduces to another polynomial h with respect to *f*, if and only if the *LT*(*g*) can be deleted by a subtraction of an appropriate multiple of the polynomial *f*. This reduction is denoted by $g \to_f h$.

Therefore, the reduction $g \to_f h$ is possible if and only if there exists a scalar *b* and a monomial *u* such that $h = g - buf$ where $b = LC(g)/LC(f)$ and $u = LM(g)/\mathrm{LM}(f)$.

A polynomial *g* reduces with respect to a set (or basis) of polynomials $F = \{f_1, f_2, \ldots, f_s\}$ if *g* is reducible with respect to one or more polynomials in *F*. In this case the reduction of one polynomial can lead to a whole sequence of reductions, which has to end after a finite number of reductions. It also can be shown that the subtraction of each polynomial $g_i$ in the sequence of reduction and the polynomial *g* itself is an element of the ideal $(f_1, f_2, \ldots, f_s)$.

The polynomial $g_i$, which is obtained after applying an i-times reduction to the polynomial g, is called the normal form respect to a set of polynomials *F*.

**S-polynomials**

This leads to another type of polynomial. These are called the S-polynomial. For two polynomials *f* and *g* the *S-polynomial* is defined:

$$S(f,g) = \frac{x^\gamma}{LT(f)} \cdot f - \frac{x^\gamma}{LT(g)} \cdot g \tag{15}$$

Where $x^{\gamma}$ denotes the largest common monomial of the leading monomial of the two polynomials $f$ and $g$ ( $x^{\gamma} = LMC(LM(f), LM(g))$ )

**Gröbner basis**

A Gröbner basis of a set of polynomial is a special basis of their ideal which has the property that:

1. every polynomial in the ideal reduces to 0 with respect to the basis,
2. every polynomial has a unique normal form with respect to the basis.

If is defined a monomial ordering. Final set $G = \{g_1, \ldots, g_t\}$ of ideal I is a Gröbner basis (or standard basis), if

$$\langle LT(g_1), \ldots, LT(g_t) \rangle = \langle LT(I) \rangle . \quad \textbf{(16)}$$

We can say that the set $\{g_1, \ldots, g_t\} \subset I$ is the Gröbner basis of I if and only if the leading term of arbitrary element from $I$ is divisible $LT(g_i)$ for any $i$.

In the first case, where parameterization entering like polynomials can be polynomial representation expressed in a form

$$\begin{aligned} x_1 &= f_1(t_1, \ldots, t_m), \\ &\vdots \\ x_n &= f_n(t_1, \ldots, t_m), \end{aligned} \quad \textbf{(17)}$$

where $f_1, \ldots, f_n$ are polynomials from $k[t_1, \ldots, t_m]$ (where $k$ is an arbitrary field). System (Eq. (17)) is projection $F : k^m \to k^n$ defined by

$$F(t_1, \ldots, t_m) = (f_1(t_1, \ldots, t_m), \ldots, (f_n(t_1, \ldots, t_m))) . \quad \textbf{(18)}$$

Then $F(k^m) \subset k^n$ is a subset $k^n$ parameterized by Eq. (17). Since $F(k^m)$ don't must be affine variety. Solution of the conversion problem from parametric description to implicit description is to find minimal variety, which contains $F(k^m)$. So implicitization is elimination of parameters from parametric description (Eq. (17)). Final equation contains only variables $x_1, \ldots, x_n$. Variables elimination can be done by a calculation of reduced Gröbner basis for an ideal $I = \langle x_1 - f_1, \ldots, x_n - f_n \rangle$. For this cope only competent selection of ordering variables.

The second way is a *rational implicitization*. The rational implicitization can be generally expressed in a form

$$
\begin{aligned}
x_1 &= \frac{f_1(t_1,\ldots,t_m)}{g_1(t_1,\ldots,t_m)},\\
&\vdots\\
x_n &= \frac{f_n(t_1,\ldots,t_m)}{g_n(t_1,\ldots,t_m)},
\end{aligned}
\qquad \textbf{(19)}
$$

where $f_1, g_1, \ldots, f_n, g_n$ are polynomials from $k[t_1,\ldots,t_m]$. Projection $F: k^m \to k^n$ can not be defined at full $k^m$, because it is necessary to exclude from $k^m$ points $(t_1,\ldots,t_m)$ for which $g_i(t_1,\ldots,t_m) = 0$ for any *i*. If we denote $W = V(g_1,\ldots,g_n) \subset k^m$, then

$$
F(t_1,\ldots,t_m) = \left( \frac{f_1(t_1,\ldots,t_m)}{g_1(t_1,\ldots,t_m)},\ldots, \frac{f_n(t_1,\ldots,t_m)}{g_n(t_1,\ldots,t_m)} \right) \qquad \textbf{(20)}
$$

defines projection $F: k^m - W \to k^n$. The goal is to find the minimal variety in $k^n$ including $F(k^m - W)$. In the defined parameterization must be eliminated fractions by multiply i[th] equation by the function $g_i$. Then the equation $1 - g_1 \ldots g_n y = 0$ for nonzero $g_1,\ldots,g_n$ on the defined variety is added and the reduced Gröbner base evaluated. Elements of the Gröbner basis which does not contain variables $y, t_i$, define the implicit representation of the given affine variety.

Gröbner basis was the part of complex mathematical expression and it is used for the transformation of a parametric description of an affine variety to the implicit representation. More information and the definitions necessary for detail understanding of this method are in [Bastl01], [Berchtold00] or [Hoffmann93]. Note, that Gröbner base of an ideal can be used also for automatic proving in geometry or robotics. A lot of commercial and noncommercial packages for the Gröbner basis solution exist. In the example below is showed the usage of Gröbner basis for finding implicit representation of torus.

**Example**

Parametric expression of torus:

$$
\begin{aligned}
x &= r\cos u\cos t + R\cos t\\
y &= r\cos u\sin t + R\sin t\\
z &= r\sin u
\end{aligned}
\qquad \textbf{(21)}
$$

adopt marking

$$
c_u = \cos u, \qquad c_t = \cos t, \qquad s_u = \sin u, \qquad s_t = \sin t \qquad \textbf{(22)}
$$

then polynomials in variables $c_u, s_u, c_t, s_t, x, y, z$ are

$$\begin{aligned} x - rc_u c_t - Rc_t &= 0 \\ y - rc_u s_t - Rs_t &= 0 \\ z - rs_u &= 0 \end{aligned} \tag{23}$$

adding identity

$$\begin{aligned} \cos^2 u + \sin^2 u = 1 \quad &\leftrightarrow \quad c_u^2 + s_u^2 - 1 = 0 \\ \cos^2 t + \sin^2 t = 1 \quad &\leftrightarrow \quad c_t^2 + s_t^2 - 1 = 0 \end{aligned} \tag{24}$$

Reduced Gröbner basis for ideal I generated by the polynomials (22) and (23) contains 9 elements. Only one from these 9 elements does not contain any variable from $c_u, c_t, s_u, s_t$ and has form

$$\begin{aligned} x^4 + 2x^2y^2 + 2x^2z^2 - (2R^2 + 2r^2)x^2 + y^4 + 2y^2z^2 - (2R^2 + 2r^2)y^2 + \\ + z^4 - (2r^2 - 2R^2)z^2 + r^4 - 2r^2R^2 + R^4 = 0 \end{aligned} \tag{25}$$

after some operations the form is

$$(x^2 + y^2 + z^2 - r^2 - R^2)^2 = 4R^2(z^2 - r^2) \tag{26}$$

and it is implicit equation of torus.

### 2.1.2 Resultant method

Term resultant is generally introduced if the question is explored: When have two polynomials in $k[x]$ common divider? A resultant is a characteristic projection variety of defined polynomial set to the smaller set of variables. Methods for a resultant evaluation can be used for an elimination of some variables subset from starting system of nonlinear algebraic equations. Interested feature of a resultant for polynomials in more variables is that from $n+1$ polynomials it eliminates *n* variables concurrently. The elimination process is not sequential like in case of Gröbner basis. The basic idea of multidimensional resultants is conversion of nonlinear elimination problem to the linear. This help to apply knowledge of linear algebra and methods for linear equation sets solving.

There exist several types of resultant. The basic definition of resultants is for two polynomials in one variable. It is for example Sylvester's or Bezout's resultant. Then generalization from two polynomials in one variable to resultants for two polynomials in two variables and to resultants for three polynomials in two variables (Dixon's resultant) is performed. Dixon's resultant can be generalized for case of $n+1$ polynomials in *n* variables. Only the Sylvester's resultant and some notes for the Bezout's resultant and Dixon's resultant will be given further. More information can be found in [Bastl03] or [Berchtold00].

**Sylvester's resultant**

The main problem is a tendency to find if two polynomials $g, f \in k[x]$ have common divider. There exist several possible ways how we can find it. For example Euclid's

algorithm can be used or a decomposition of polynomials to product root factors. Or there is lemma too, which says if $g, f \in k[x]$ and $\deg(f) = n > 0$, $\deg(g) = m > 0$ then $f$ and $g$ have common divider if and only if exist polynomials $A, B \in k[x]$ such that:

1. Both polynomials A and B are not equal to zero.
2. A is at most degree $m-1$ and B is at most degree $n-1$
3. $Af + Bg = 0$

Equation $Af + Bg = 0$ may be re-written to the linear equation set for unknown coefficients polynomial A and B. Note, that elements of the matrix depend on coefficients of polynomial $f$ and $g$. Sylvester's resultant is defined by the next definition.

*Definition*

Let $g, f \in k[x]$ be polynomials of positive degree in a form

$$f = a_n x^n + \cdots + a_0, \qquad a_n \neq 0$$
$$g = b_m x^m + \cdots + b_0, \qquad b_m \neq 0$$

then *Sylvester's matrix* of polynomials $f$ and $g$ is of type $(n+m)\times(n+m)$ and form

$$\mathrm{Syl}(f,g) = \underbrace{\begin{pmatrix}
a_n & & & & b_m & & & \\
a_{n-1} & a_n & & & b_{m-1} & b_m & & \\
a_{n-2} & a_{n-1} & \ddots & & b_{m-2} & b_{m-1} & \ddots & \\
\vdots & \vdots & & a_n & \vdots & \vdots & & b_m \\
a_0 & & & a_{n-1} & b_0 & & & b_{m-1} \\
 & a_0 & & \vdots & & b_0 & & \vdots \\
 & & \ddots & \vdots & & & \ddots & \vdots \\
 & & & a_0 & & & & b_0
\end{pmatrix}}_{n+m},$$

Empty places represent zeros. *Sylvester's resultant* of polynomials $f$ and $g$ is then determinant of Sylvester's matrix and is denoted **Res(*f*,*g*)** also

$$\mathrm{Res}(f,g) = \det(\mathrm{Syl}(f,g))$$

To consider Sylvester's resultant definition, it can be observed that $f$ and $g$ have common divider if and only if Res($f$,$g$) = 0.
Next example shows demonstration of Sylvester's resultant and comparison with solution of the same set using Gröbner basis of ideal method.

*Example*

We have polynomials

$$f = x^2 y - 1$$
$$g = x^2 + y^2 + xy - 4.$$

They are polynomials in variable $x$ whose coefficients are polynomials in variable $y$. Sylvester's resultant for this polynomials is

$$\mathrm{Res}(f,g,x) = \det\begin{bmatrix} y & 0 & 1 & 0 \\ 0 & y & y & 1 \\ -1 & 0 & y^2 - 4 & y \\ 0 & -1 & 0 & y^2 - 4 \end{bmatrix} = y^6 - 8y^4 + y^3 + 16y^2 - 8y + 1.$$

For comparison can be showed solution of the same set of polynomials using the Gröbner basis of ideal. For the ideal $I = \langle f, g \rangle$, reduced Gröbner basis is

$$I = \left\langle x - 4y^5 - y^4 + 32y^3 + 4y^2 - 64y + 16, y^6 - 8y^4 + y^3 + 16y^2 - 8y + 1 \right\rangle.$$

It can be seen that resultant directly corresponds to the element of the eliminated ideal I.

**Bezout's resultant**

It is similar to the Sylvester's resultant that it is defined by matrix (Bezout's matrix), which has defined properties. The creation if Bezout's matrix is more difficult then the creation Sylvester's matrix but finally the Bezout's matrix is much smaller (n × n). The determinant evaluation from the Bezout's matrix is therefore much more faster. The Bezout's matrix can be obtained from the Sylvester's matrix by the special transformation.

**Dixon's resultant**

It is a generalization of Bezout's matrix and Bezout's resultant for three polynomials in two variables. Dixon's resultant is then generalized to $n+1$ polynomials in $n$ variables.

### 2.1.3 The Wu-Ritt method

This section gives a brief introduction to the theory of this method. This method is based on Wu-Ritt's approach to find a characteristic set for a nonlinear system of equation. A given system of polynomial equations $S = \{f_1, f_2, \ldots, f_m\}$ is transformed into a triangular form S'. It is important to note that if the number $n$ of variables is greater then the number of equations in a set $S$ ($n > m$) then the variable set is divided into two subsets: the independent variables (denoted by $u_1, \ldots, u_k$) and the dependent variables (denoted by $y_1, \ldots, y_l$).

Pseudo division of two multivariate polynomials is the key operation in characteristic set computation. To perform the pseudo division, the recursive representation of a polynomial, which is considered as a univariate polynomial in its highest variable, is used. This pseudo division defines a polynomial reduction.

A polynomial $f_i$ is reduced with respect to another polynomial $f_j$ if

1. the highest variable of $f_i$ is $\prec$ the highest variable of $f_j$ or
2. the degree of the highest variable in $f_j$ is greather than the degree of the highest variable in $f_i$.

If $f_i$ is not reduced with respect to $f_j$ then $f_i$ reduces to $r$ by pseudo-dividing by $f_j$.

A characteristic set $\Phi$ is defined:
Given a finite set $\Sigma$ of polynomials in $u_1,\ldots,u_k,y_1,\ldots,y_l$, a characteristic set $\Phi$ of $\Sigma$ is defined to be either

1. $\{g_1\}$ where $g_1$ is a polynomial in $u_1,\ldots,u_k$ or
2. a chain $\langle g_1 \ldots g_l \rangle$, where $g_1$ is a polynomial in $y_1,u_1,\ldots,u_k$ with $LC(g_1)$, $g_2$ is a polynomial in $y_2,y_1,u_1,\ldots,u_k$ with $LC(g_2)$, …, $g_l$ is a polynomial in $y_l,\ldots,y_1,u_1,\ldots,u_k$ with $LC(g_l)$, such that
   - any zero of $\Sigma$ is zero of $\Phi$, and
   - any zero of $\Phi$ that is not a zero of any of the leading coefficients $LC(g_i)$ is a zero of $\Sigma$.

The optimal algorithm for a characteristic set computation is in [Gallo90]. In this paper, parallel and sequential algorithm is introduced. The time complexity of the sequential algorithm is $O(N^{2.376})$ and for the parallel algorithm, time complexity is $O(\log^2 N)$.

More information about this method and algorithms for the characteristic set solution are in [Gallo90], [Gallo91], [Berchtold00] or [Rege96]. The Characteristic Sets Method has been implemented on most Computer Algebra Systems including *Mathematica, Maple, Macsyma, Axiom* etc.

All methods mentioned above haves one common property: if we want to use them to convert parametric description of object to the implicit definition, the output from these methods is equation. From each method described above we receive implicit equation and this implicit equation can be directly used in visualizations methods. These are methods of geometric modeling. Slightly different methods can be used in computer graphics, too. There is no need to know parametric description of the object. These methods stem from knowledge triangular mesh or vertex data set.

The methods described above do not will be more analyze and use in further work. This was only briefly description of these methods. For more details search in publications referred in text.

## 2.2 Variational implicit surfaces (RBF method)

If an object is defined by the implicit equation it is a perfect description of the object. The object can be directly visualized by any method (1.4) from the implicit form. The object can be also stored in its implicit form for later use. So it is good to have the object defined by the implicit equation. A lot of objects, especially the basic primitives for the CSG modeling, are described by the implicit equation. Sometimes we want to use the object model, which has no implicit description. There will be described an elaborated method in this chapter. This method is based on *Variational implicit surfaces*, for the implicit representation of the object surface.

### 2.2.1 Problem definition

The surface representation problem can be expressed as

Problem:

Given $n$ distinct points $x_1, x_2, \ldots, x_n$ on a surface $S$ in $\Re^3$, find a surface $S$' that is a reasonable approximation to $S$.

Our approach is to model the surface implicitly with a function $f(x, y, z)$. If a surface $S$ consists of all the points $(x, y, z)$ that satisfy the equation

$$f(x, y, z) = 0 \tag{27}$$

then we say that $f$ implicitly defines $S$. Note, that the object can be defined like point-cloud data or triangular mesh. If the object is defined like the triangular mesh it helps with a set equation definition, but about it later.

### 2.2.2 Scattered data interpolation

The shape transformation problem relies on *scattered data interpolation*. The problem of scattered interpolation is to create a smooth function that passes through a given set of data points. The two-dimensional version of this problem can be stated as follows: Given a collection of $k$ constraint points $\{c_1, c_2, \ldots, c_k\}$[2] that are scattered in the plane, together with scalar height values at each of these points $\{h_1, h_2, \ldots, h_k\}$, construct a smooth surface that matches each of these heights at the given locations. We can think of this solution surface as a scalar-valued function $f(\mathbf{x})$ so that $f(\mathbf{c}_i) = h_i$, for $1 \le i \le k$. One common approach to solve scattered data problems is to use variational techniques. This approach begins with an energy that measures the quality of an interpolating function and then finds the single function that matches the given data points and that minimizes this energy measure. For two-dimensional problems, thin-plate interpolation is the variational solution when using the following energy function $E$:

$$E = \int_\Omega f_{xx}^2(x) + 2f_{xy}^2(x) + f_{yy}^2(x) \tag{28}$$

[2] The constraints points contain all points of the object and additional points. How can be defined additional points will be discused later.

The notation $f_{xx}$ means the second partial derivative in the *x* direction, and the other two terms are similar partial derivatives, one of them mixed. The above energy function is basically a measure of the aggregate squared curvature of $f(\mathbf{x})$ over the region of interest $\Omega$. Any creases or pinches in a surface will result in a larger value of *E*. A smooth surface that has no such regions of high curvature will have a lower value of *E*. The thin-plate solution to an interpolation problem is the function $f(\mathbf{x})$ that satisfies all of the constraints and that has the smallest possible value of *E* [Turk99].

The scattered data interpolation problem can be formulated in any number of dimensions. When the given points $\mathbf{c}_i$ are positions in *N*-dimensions rather than in 2-d, this is called the *N*-dimensional scattered data interpolation problem. There are appropriate generalizations to the energy function and to thin-plate interpolation for other dimensions. Because the term *thin-plate* is only meaningful for 2D problems, we will use *variational interpolation* to mean the generalization of thin-plate techniques to any number of dimensions.

### 2.2.3 Equation system

Now we have definition of the problem and we can describe the solution of a variational problem. The scattered data interpolation task as formulated above is a variational problem where the desired solution is a function, $f(\mathbf{x})$, that will minimize equation (28) subject to the interpolation constraints $f(\mathbf{c}_i) = h_i$. Equation (28) can be solved using weighted sums of the radial basis function $\phi(x)$.

Scattered data interpolation can be achieved using *radial basis functions* centered at the constraints. Radial basis functions are circularly symmetric functions centered at a particular point. Radial basis functions may be used to interpolate a function with *n* points by using *n* radial basis functions centered at these points. The resulting interpolated function thus becomes

$$f(x) = \sum_{j=1}^{n} \lambda_j \phi\left(\left\| x - c_j \right\|\right). \tag{29}$$

In the above equation, $c_j$ are the locations of the constraints, $\lambda_j$ are the *weights* and $\phi$ is a *radial basis function* evaluated in radial *r* defined by the difference of the point in which we want to evaluate this function and the constraints.

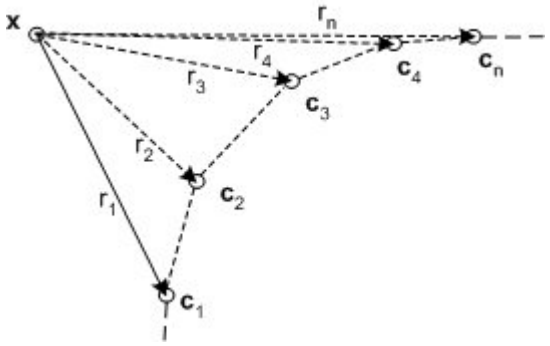



**Figure 7: Solution of equation (29) for an arbitrary point x.**

Figure 7 shows how the scalar value for an arbitrary point x is evaluated. To solve for the set of $\lambda_j$ that will satisfy the interpolation constraints $h_i = f(c_i)$, we can substitute the right side of equation (29) for $f(c_i)$, which gives:

$$f(c_i) = \sum_{j=1}^{n} \lambda_j \phi\left(\left\| c_i - c_j \right\|\right) = h_i \tag{30}$$

Since this equation is linear with respect to the unknowns $\lambda_j$, it can be formulated as a linear system. For interpolation in 3-d space, let $c_i = \{c_i^x, c_i^y, c_i^z\}$ and let $\phi_{ij} = \phi(c_i - c_j)$. Then this linear system can be written as follows:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} \tag{31}$$

In some cases (including the thin-plate spline solution), it is necessary to add a first-degree polynomial $P$ to account for linear and constant portion of $f$ and ensure positive-definiteness of the solution. Then equation (29) is modified to equation (32).

$$f(x) = \sum_{j=1}^{n} \lambda_j \phi\left(\left\| x - c_j \right\|\right) + P(x) \tag{32}$$

If a polynomial is required, Eq. (31) similarly becomes

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \cdots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \cdots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \cdots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{33}$$

If we denote

$$\begin{aligned} A_{i,j} &= \phi\left(\left\| c_i - c_j \right\|\right), && i, j = 1, \ldots, n \\ P_{i,j} &= c_j(x_i), && i = 1, \ldots, N, j = 1, \ldots, n \end{aligned} \tag{34}$$

then we can write the equation system in a form

$$\begin{bmatrix} A & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ p \end{bmatrix} = B \begin{bmatrix} \lambda \\ p \end{bmatrix} = \begin{bmatrix} h \\ 0 \end{bmatrix} \tag{35}$$

It can be seen that in this system of equations, the part for solving of the coefficients polynomial $\left(p^x, p^y, p^z\right)$ was added. Notation of writing of the polynomial to the system can be deducted from the matrix notation for the equation of plane determined by three points. These points cannot lie on the line.

The system (33) must be solved and then the equation (29) can be evaluated for an arbitrary point in defined spatial coordinates. Discussion about solving the system and setting variables of the system is in next chapters.

### 2.2.4 Solving the system

The matrix system and solvability of this system must be examined before the discussion about methods for solving linear system. Certain that

$$f(c_i^x, c_i^y, c_i^z) = 0, \qquad i = 1, \ldots, n \quad \text{(on-surface points)} \tag{36}$$

If basis function $\phi$ is known then we know the values of all elements of the matrix *B*. It is obvious, that condition $h_i = 0$ is satisfied for on-surface points thus *h* is zero vector. So it can be easily seen, that it is possible to create the homogeneous system (37), from which it is possible to calculate values of vectors $\lambda$ and *p*.

$$B \begin{bmatrix} \lambda \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{37}$$

This system consisting of *n* (*n* = *n*+N+1) homogeneous equations has always zero solution $0 = (0,0,\ldots,0)$. If the matrix of the homogeneous system has rank *h*, then the system has (*n*-*h*) linear independent solutions and each solution of this system is a linear combination of these (*n*-*h*) solutions. Especially if $h = n$, the system has only zero solution $0 = (0,0,\ldots,0)$. If $m = n$ (number of rows is equal to number of columns) then the system has nonzero solution if and only if the determinant of the system is equal to zero. The system matrix *B* has at the main diagonal zero elements and out of the main diagonal has nonzero elements. It can be verified that the rank of our system matrix is equal to *n* (definition below) and its determinant is nonzero.

***Definition***

The square matrix $A = (a_{ij})$ of rank *n* is regular if and only if its determinant $\left|a_{ij}\right|$ is nonzero. (i.e. the matrix *A* has rank *n*) [Rektorys95].

Now we can say that our system has only one solution, zero (trivial) solution. This exact solution of the system is not what we need. This solution cannot be used for visualization methods. We need an approximation of such solution, thus it is necessary to add perturbation to input data to get non zero solution. This perturbation adds new equations to the system, so the system is no longer homogeneous. See next section about perturbation in data.

LU factorization [Turk99], [Turk01], [Turk02], [Morse01] or [Yngve02] is the mostly used method for solving the linear system defined by the equation (33) and (31). This method comes from rule that $LU = A$. The system matrix is decomposed to the upper triangular ($U$ matrix) and to the lower triangular ($L$ matrix) with unity elements at the diagonal. It is possible for such matrix decomposition to state following:

$$\begin{aligned} Ax &= LUx = b \\ Ly &= b \\ Ux &= b. \end{aligned} \tag{38}$$

Advantage of the LU factorization is that we can solve easily more systems of the equations with equal system matrix ($A$) but different right sides (vector $b$). Factorization can be solved without knowledge right side. Then the total number of operations for LU factorization is $n^3$ for such case. If vector $b$ (right side) is known, then there are only $n^2$ operations needed for LU factorization. Methods like Cholesky factorization [Beatson00] or GMRES iterative method [Beatson99] can be also used.

### 2.2.5 Perturbation in data

In order to avoid the trivial solution that $f$ is zero everywhere, off-surface points are appended to the input data and are given non-zero values. This gives a more useful interpolation problem: Find $f$ such that

$$\begin{aligned} f(c_i^x, c_i^y, c_i^z) &= 0, \qquad i = 1, \ldots, n \quad \text{(on-surface points)}, \\ f(c_i^x, c_i^y, c_i^z) &= d_i \neq 0, \qquad i = n+1, \ldots, N \quad \text{(off-surface points)}. \end{aligned} \tag{39}$$

This still leaves the problem of generating the off-surface points $\{(x_i, y_i, z_i)\}_{i=n+1}^{N}$ and the corresponding values $d_i$.

An obvious choice for $f$ is a *signed-distance function*, where the $d_i$ are chosen to be the distance to the closest on-surface point. Points outside the object are assigned positive values, while points inside are assigned negative values. Similar to Turk & O'Brien [Turk02], these off-surface points are generated by projecting along surface normals. Off-surface points may be assigned to either side of the surface as illustrated in Figure 8

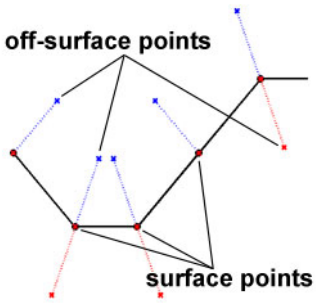



**Figure 8: A signed-distance function is constructed from the surface data by specifying off-surface points along surface normals. These points may be specified on either or both sides of the surface, or not at all.**

Experience has shown that it is better to augment a data point with two off-surface points, on either side of the surface. In Figure 9a, surface points from a laser scan of a hand are

shown in green. Off-surface points are color coded according to their distance from their associated on-surface point. Hot colors (red) represent positive points outside the surface while cold colors (blue) lie inside. It is different definition from ours: we have positive values inside and negative values outside (3). There are two problems to solve; estimating surface normals and determining the appropriate projection distance.

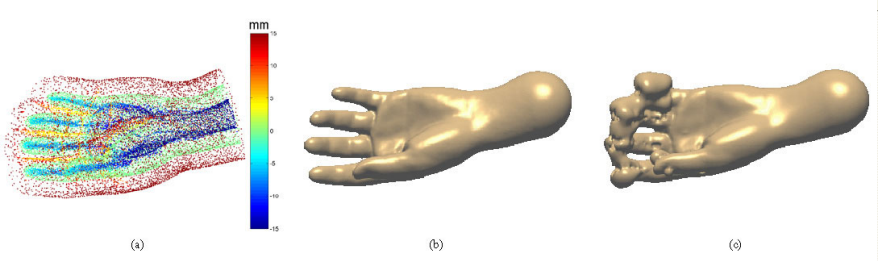



**Figure 9: Reconstruction of a hand from a cloud of points with and without validation of normal lengths. [Carr01]**

If we have a partial mesh, then it is straightforward to define off-surface points since normals are implied by the mesh connectivity at each vertex. In the case of unorganized point-cloud data, normals may be estimated from a local neighbourhood of points. This requires estimating both the normal direction and determining the sense of the normal. We can locally approximate the point-cloud data with a plane to estimate the normal direction and use consistency and/or additional information such as scanner position to resolve the sense of the normal. In general, it is difficult to robustly estimate normals everywhere. However, unlike other methods [Hoppe92], which also rely on forming a signed-distance function, it is not critical to estimate normals everywhere. If normal direction or sense is ambiguous at a particular point then we do not fit to a normal at that point. Instead, we let the fact that the data point is a zero-point (lies on the surface) to tie down the function in that region.

Given a set of surface normals, care must be taken when projecting off-surface points along the normals to ensure that they do not intersect other parts of the surface. The projected point is constructed so that the closest surface point is the surface point that generated it. Provided this constraint is satisfied, the reconstructed surface is relatively insensitive to the projection distance $|d_i|$. Figure 9c illustrates the effect of projecting off-surface points with inappropriate distances along normals. Off-surface points have been chosen to lie a fixed distance from the surface. The resulting surface, where $f$ is zero, is distorted in the vicinity of the fingers where opposing normal vectors have intersected and generated off-surface points with incorrect distance-to-surface values, both in sign and magnitude. In Figure 9a and b, validation of off-surface distances and dynamic projection has ensured that off-surface points produce a distance field consistent with the surface data.

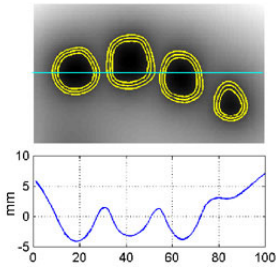
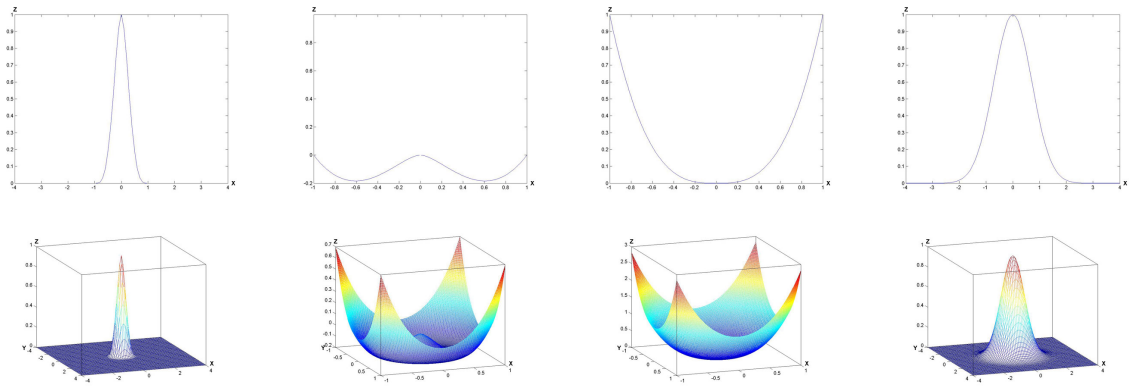



**Figure 10: Cross section trough the fingers of a hand reconstructed from the point-cloud in Figure 9. [Carr01]**

Figure 10 is a cross section through the fingers of the hand. The figure illustrates how the RBF function approximates a distance function near the object's surface. The approximately equally spaced iso-contours at +1, 0 and -1 in the top of the figure and the corresponding function profile below, illustrate how the off-surface points have generated a function with a gradient magnitude close to 1 near the surface (which corresponds to the zero-crossings in the profile shown).

### 2.2.6 Basic function

The choice of basic function $\phi$ affects the form of the attached surface. There exist many different functions, which can be used. Popular choices for the basic function include the thin-plate spline $\phi(r) = r^2 \log(r)$ (for fitting smooth functions of two variables), the Gaussian $\phi(r) = \exp(-cr^2)$ (mainly for neural networks), and the multiquadric $\phi(r) = \sqrt{r^2 + c^2}$ (for various applications, in particular fitting to topographical data). For fitting functions of three variables, good choices include the biharmonic ($\phi(r) = r^2$) and triharmonic ($\phi(r) = r^3$) splines.



**a) Compactly Supported** $\phi(r) = (1-r)_+^4(4r+1)$

**b) Thin-plate (2-d)** $\phi(r) = r^2 \log r$

**c) Thin-plate (3-d)** $\phi(r) = r^3$

**d) Gaussian** $\phi(r) = e^{r^2\sigma^2}$

**Figure 11: Comparison of different basic functions in 2-d and 3-d.**

On initial inspection, the essentially local nature of the Gaussian, inverse multiquadric ($\phi(r) = (r^2 + c^2)^{-1/2}$) and compactly supported basic functions appear to lead to more desirable properties in the RBF. For example, the matrix B now has special structure (sparsity), which can be exploited by well-known methods, and evaluation of Equation (29) only requires that the sum be over nearby centers instead of all *N* centers. However, non-compactly supported basic functions are better suited to extrapolation and interpolation of irregular, non-uniformly sampled data. Indeed, numerical experiments using Gaussian and compactly supported piecewise polynomials for fitting surfaces to point-clouds have shown that these basic functions yield surfaces with many undesirable artifacts in addition to the lack of extrapolation across holes.

### Compactly-Supported Radial Basis Functions

Wendland [Wendland95] has recently solved the minimum degree polynomial problem with compact, locally supported radial basis functions that guarantee positive-definiteness of the matrix [Figure 11a]. All of the solutions have the form

$$\phi(r) = \begin{cases} (1-r)^p P(r) & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases} \quad \textbf{(40)}$$

For various degrees of desired continuity ($C^k$) and dimensionality ($d$) of the interpolated function, he has derived the following radial basis functions:

| | | |
|---|---|---|
| d = 1 | $(1-r)_+$ | $C^0$ |
| | $(1-r)_+^3(3r+1)$ | $C^2$ |
| | $(1-r)_+^5(8r^2+5r+1)$ | $C^4$ |
| d = 3 | $(1-r)_+^2$ | $C^0$ |
| | $(1-r)_+^4(4r+1)$ | $C^2$ |
| | $(1-r)_+^6(35r^2+18r+3)$ | $C^6$ |
| | $(1-r)_+^8(32r^3+25r^2+8r+1)$ | $C^6$ |
| d = 5 | $(1-r)_+^3$ | $C^0$ |
| | $(1-r)_+^3(5r+1)$ | $C^2$ |
| | $(1-r)_+^7(16r^2+7r+1)$ | $C^4$ |

These functions have radius of support equal to 1. Scaling of the basis functions (i.e., $\phi(r/\alpha)$ ) allows any desired radius of support $\alpha$.

The radial basis functions have finite support, $\phi(\| c_i - c_j \|) = 0$ for all ($c_i$, $c_j$) farther apart than the radius of support. We can exploit the spatial locality of the compactly supported radial basis functions during evaluation of the embedding function $f$ by recognizing that only a fraction of the terms of Eq. (29) are non-zero for a given $x$: $\phi(\| c_i - c_j \|) \neq 0$ if and only if $\| c_i - c_j \| < 1$. By again using a $k$-d tree [More01] to organize the constraints spatially, each evaluation of the interpolating function requires only $O(\log n)$ operations to determine these non-zero terms.

**Selecting the Radius of Support**

The finite radius of support introduces an additional parameter that doesn't exist in the thin-plate implementation. Proper selection of the radius of support is critical to achieve optimal efficiency of computation and results. Too small radius can produce basis functions that are unable to span the inter-constraint gaps. Too large radius does not adversely affect the results but reduces the sparseness of the matrix, thus increasing the computation required. It is therefore necessary to select a radius of support that is both large enough to produce effective results and not so large that the computation becomes impractical.

### 2.2.7 Speedup techniques

The RBF method is computational very expensive. There exist many different ways to speedup this method. Some of them are based on preprocessing such as reducing the start point set and other on the speedup of methods for linear system equation evaluation. In this section some of them will be introduced.

#### Fast Multipole Method

The Fast Multipole Method (FMM) was introduced in [Greengard87] and for speedup of the RBF evaluation is used by Carr et al. [Carr01]. The FMM was designed for the fast evaluation of potentials (harmonic RBF's) in two and three dimensions. The FMM makes use of the simple fact that when computations are performed, infinite precision is neither required nor expected. Once this is realized, the use of approximations is allowed. For the evaluation of an RBF, the approximations of choice are far- and near-field expansions. With the centers clustered in a hierarchical manner, far- and near-field expansions are used to generate an approximation to that part of the RBF due to the centers in a particular cluster. A judicious use of approximate evaluation for clusters "far" from an evaluation point and direct evaluation for clusters "near" to an evaluation point allows the RBF to be computed to any predetermined accuracy and with a significant decrease in computation time compared with direct evaluation.

#### Center reduction

Conventionally, an RBF approximation uses all the input data points as nodes of interpolation, and as centers of the RBF. However, the same input data may be able to be approximated to the desired accuracy using significantly fewer centers, as illustrated in Figure 12. A greedy algorithm can therefore be used to iteratively fit an RBF to within the desired *fitting accuracy*.

A simple greedy algorithm consists of the following steps:

1. Choose a subset from the interpolation nodes $x_i$ and fit an RBF only to these.
2. Evaluate the residual, $\varepsilon_i = f_i - f(x_i)$, at all nodes.
3. If $\max\{|\varepsilon_i|\} <$ *fitting accuracy* then => stop.
4. Else append new centers where $\varepsilon_i$ is large.
5. Re-fit RBF => goto 2.

If a different accuracy $\varepsilon_i$ is specified at each point, then the condition in step 3 may be replaced by $|\varepsilon_i| < \delta_i$.

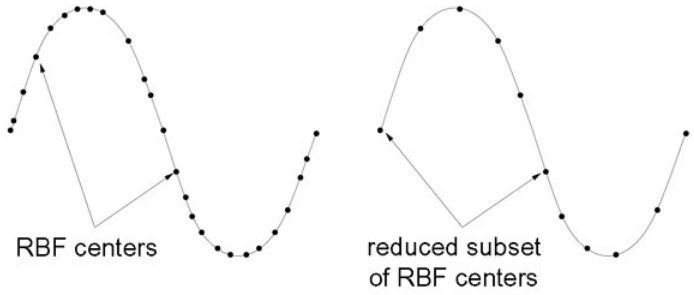



**Figure 12: Illustration of center reduction. [Carr01]**

Center reduction is not essential when using the fast methods described in previous section. Usage of this method can be found in [Carr01]. Methods for polygonal mesh reduction [Franc02] can be used, too.

**Subset choosing**

The subset from the main set of points is chosen mainly together with compactly supported radial basis functions [Morse01]. The compactly supported RBF have defined radius of support and $\phi(x) = 0$ for all $x$ farther than the radius. By using a $k$-d tree, the set of all points within the distance $r$ from the particular point $c_i$ can be determined in $O(\log n)$ time. A $k$-d tree is a multidimensional binary tree with the following sorting property for a tree with point $x$ at the root and subtrees $T_{\text{left}}$ and $T_{\text{right}}$.

$$\begin{aligned} \forall y \in T_{left} : y^d \le x^d \\ \forall y \in T_{right} : y^d > x^d \end{aligned} \qquad \textbf{(41)}$$

where the sorting dimension $d$ changes at each level of the tree. $k$-d trees can be used to find all points within distance $r$ of a particular constraint in $O(n \log n)$ time.

The resulting matrix is extremely sparse. Using a sparse-matrix representation, only $O(n)$ storage is required. The direct (LU) sparse matrix solver can be used to find the solution to the system of equations. The computational complexity of such a solver depends on the amount of matrix "fill in" that occurs during the solution.

### Solving method

One of the speedup techniques can be selection of the method for solving the equation system. Mostly used method is LU factorization. LU factorization is used in the form for full matrix: all input points are used for construction of the equation system, or in the sparse matrix form: from the input set of points are selected only points that can affect the value of a computed point. For solving the system of equations, iterative methods like GMRES or other can be used [Mika96]. Algorithmic complexity of these methods is mentioned in the next section.

## 2.2.8 Algorithmic complexity

Calculating and using implicit surfaces that interpolate may be analyzed in three parts:

1. Constructing the system of equations,
2. Solving the system of equations, and
3. Evaluating the interpolating function (as required).

**Constructing the System of Equations**
A significant portion of the computational cost involved in calculating these implicit surfaces is the cost required to construct the matrix (or sub matrix) $\phi_{ij} = \phi(\| c_i - c_j \|)$. Recall that the thin-plate radial basis function is $\phi(r) = r^2 \log(r)$ (two dimensions) or $\phi(r) = r^3$ (three dimensions). This means that *the matrix is entirely non-zero except along the diagonal*, requiring the calculation of all inter-point distances within the set $\{c_i\}$. Although the symmetry of the matrix cuts the computational cost in half, the computational complexity is still $O(n^2)$. Furthermore, storage of such a matrix requires $O(n^2)$ floating-point values and this is a potentially more prohibitive factor than the computational complexity.

**Solving the System of Equations**
Although Turk and O'Brien use LU factorization (an $O(n^3)$ algorithm) to solve Eq. 5, they correctly point out that it is possible to solve this system in $O(n^2)$ by iterative means. Thus, while solution of the system may appear to be the limiting step, it needs only to be as computationally expensive as constructing the system. The method based on GMRES iteration method [Beatson99] reduces the computational cost of solving an RBF interpolation problem to $O$(N) storage, and $O$(N log N) operations.

**Evaluating the Function**
For nearly all applications it is not enough to simply solve the weights of the respective radial basis functions. Rather, it is necessary to evaluate this embedding function at potentially many points in order to extract the isosurface, calculate normals or other derivative quantities, etc. Because the terms $\phi(\| x - c_i \|)$ in Eq. (29) are all nonzero for the thin-plate solution (except for one zero term when $x \in \{c_j\}$), all of the terms must be used in calculating an arbitrary point. Thus, the complexity of each evaluation of the interpolated function is $O(n)$.

While the *thin-plate* spline embedding function does indeed minimize bending energy, it has the following drawbacks in computation and usefulness for user interaction:

1. $O(n^2)$ computation is required to build the system of equations.
2. $O(n^2)$ storage is required (for the nearly-full matrix) to represent the system.
3. $O(n^2)$ computation is required to solve the system of equations.
4. $O(n)$ computation is required per evaluation
5. Because every known point affects the result, a small change in even one constraint is felt throughout the entire resulting interpolated surface, an undesirable property for shape modeling.

### 2.2.9 Advantages and disadvantages

**Advantages**
A single functional description has a number of advantages over piecewise parametric surfaces and implicit patches. It can be evaluated anywhere to produce a particular mesh,

*i.e.,* a faceted surface representation can be computed at the desired resolution when required. Sparse, nonuniformly sampled surfaces can be described in a straightforward manner and the surface parameterization problem, associated with piecewise fitting of cubic spline patches, is avoided.

An RBF offers a compact functional description of a set of surface data. Interpolation and extrapolation are inherent in the functional representation. The RBF associated with a surface can be evaluated anywhere to produce a mesh at the desired resolution. The RBF representation has advantages for mesh simplification and remeshing applications. Gradients and higher derivatives are determined analytically and are continuous and smooth, depending on the choice of basic function. Surface normals are therefore reliably calculated and iso surfaces extracted from the implicit RBF model are manifold (*i.e.,* they do not self-intersect).

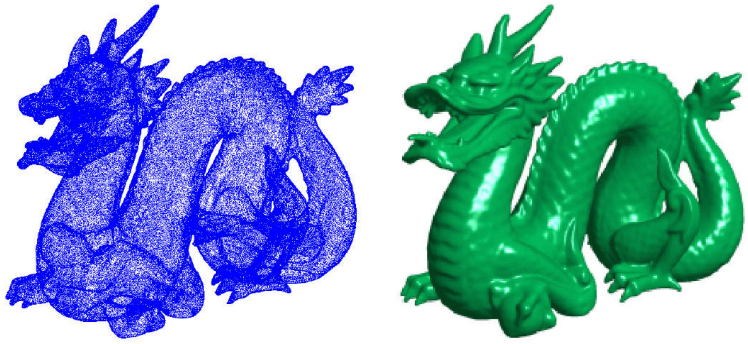


**Figure 13: Fitting a Radial Basis Function (RBF) to a 438,000 point-cloud. [Carr01]**

**Disadvantages**

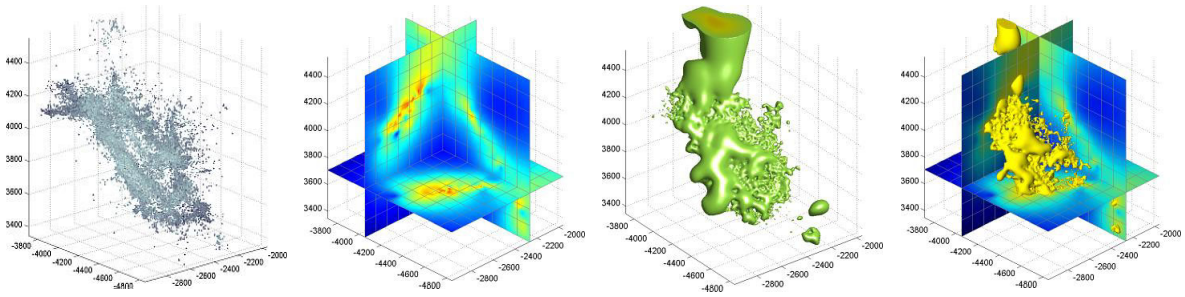
If the RBF method is used to solve, then it is necessary for the simple method without any reduction of input set of points calculate with $O(N^2)$ storage and $O(N^3)$ arithmetic operations. For example, direct fitting of the dragon in Figure 13 would have required 3,000GB just to store the corresponding matrix. Consequently, fitting RBFs to real-world scan data has not been regarded as computationally feasible for large data sets.

### 2.2.10 Examples

The RBF method is used in a lot of applications. This method is mainly used for creating implicit description of the object defined by the point-cloud data or by the polygonal mesh. The input set of points can be obtained from scientific measurement, computer tomography (CT) or magnetic resonance (MR), 3D scanners etc. Only few applications with this method will be introduced.

**Scientific visualization**

Visualization of scientific data is necessary for understanding the process and visualize to otherwise hidden structure in the data. Good example of scientific visualization can be visualization of geophysical measurements. Figure 14a shows input data set which consists of 471031 geophysical measurements in 3D.



**a) 3D view** **b) Values on planes** **c) An iso surface** **d) A combined view**

**Figure 14: Geophysical data visualization. [ARANZ]**

Images a,b,c and d in Figure 14 show multi-planar reslicing, iso surface extraction and combined view. This visualization was obtained with *FastRBF* toolbox for *MATLAB*. *Fast*RBF toolbox is product of Applied Research Associates NZ Ltd. [ARANZ].

**Medical visualization**

Visualization of medial data (CT, MR) or implant can be found in [Carr97] or [Yoo01]. The first publication [Carr97] presented a practical solution the problem of interpolating incomplete surface derived from three-dimensional medical graphics. The specific application considered is the design of cranial implants for the repair of defect, usually holes, in a skull. Terry S. Yoo [Yoo01] presents reconstruction of inner surface of blood vessel from a series of endovascular ultrasound images.

**3D scans visualization**

Many examples of 3D scans visualization introduced by J.C.Carr are in [Carr01]. Figure 9 and Figure 13 show some of the reconstructed objects. The data were obtained from 3D scanner LIDAR or CYRAX 2400. Because these data have often very many points (Dragon on Figure 13 has 438,000 points), they used center reduction and FMM for speedup of the RBF method.

## Previous work

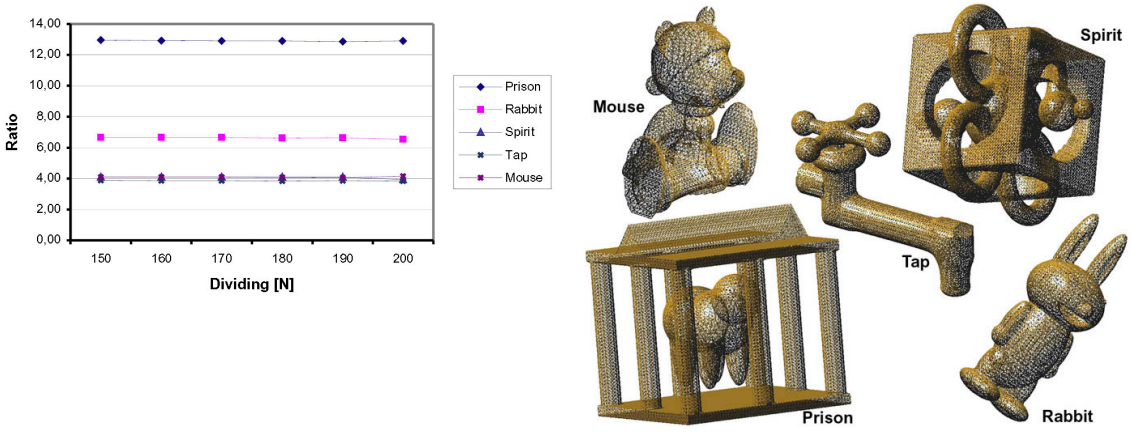
Previous work for this article has two parts. The first part consists of developing the system for direct compilation of implicitly defined objects and the second part lies in a basic implementation of radial basis functions method.

### Direct compilation

The principle of the direct compilation of implicitly defined objects was based on the simple modeling language. The modeling language was inspired by the HyperFun project [HyperFun99] and the HyperFun was the main program for comparing with our system. The goal of the system for the direct compilation of implicitly defined objects was to keep both speedup from compiled objects in standard compiler and the syntax of HyperFun language.

The syntax of the HyperFun language is simple and very similar to the C++ programming language. Some differences are only in signs, which define the Boolean operations. In particular, are speaking about the subtraction operator. In HyperFun, the backslash operator ('\') is used for this operation. It is slightly increased complexity of the model description. For other operations (union, intersection etc.), there was no problem with overloading the standard C++ operators.

The system 'Compiled HyperFun' (CHF) reached speedup from the compilation of the models. Figure 15a shows the graph of speedup ratio. Testing was made on the complex objects form [HyperFun99]. The object was rewritten to the C++ language and the speedup was tested in different programming environment (e.g. Microsoft Visual C++, Borland C++ Builder or Microsoft .NET C#). Figure 15b shows the models.




a) b)

**Figure 15: Speedup ratio and the models for testing. [Uhlir03]**

The CHF system was implemented as a module for Multivisual Environment (MVE) and it is connected with the visualization module. Both these modules are used in the implementation of the RBF method.

### RBF method

Implementation of this method is based on the complex description of the RBF method above. All tests of this method are in MATLAB v6.5 and in MVE modules noted in previous section. Note, that there exists module [ARANZ] for MATLAB that provides the

implicitization of point-cloud data or polygonal meshes. The [ARANZ] module has not been used in this project.

The RBF method was implemented without any special speedup technique. For solving the linear equation system, LU factorization was used. Because the RBF method is very computationally expensive and does not use any method for the center reduction, then we tested the method for few points only.

## Example A

We have few points, which define plane in 3-d space [Figure 16].

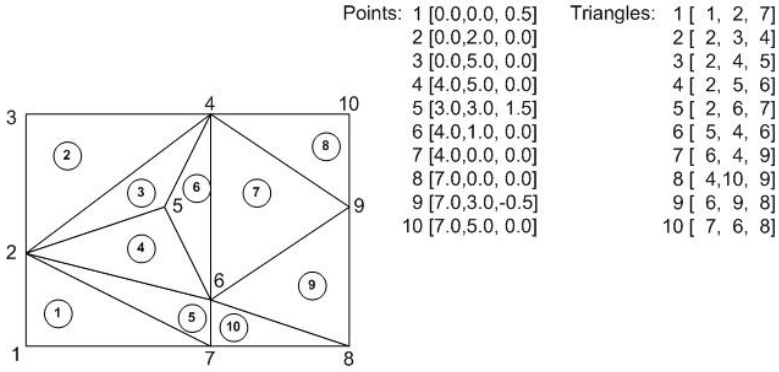



**Figure 16: Plane definition.**

This plane can be visualized in MATLAB with functions *trimesh* or *trisurf*. Figure 17 shows the plane in 2-d view and 3-d view.

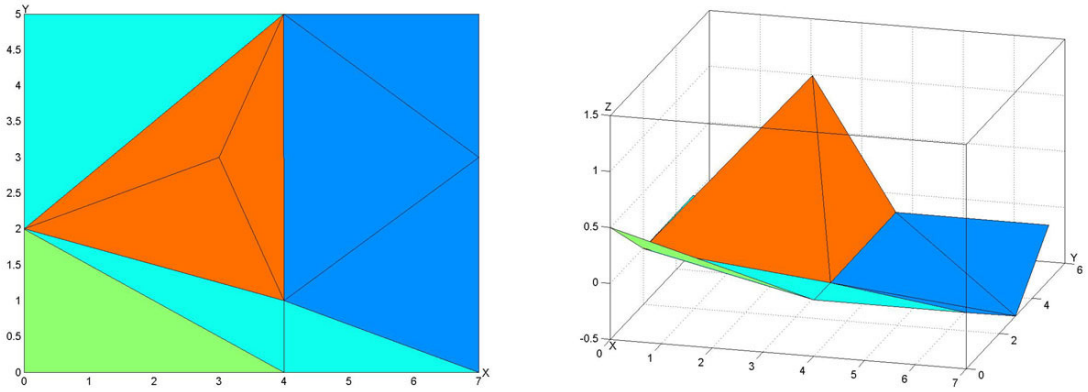


**Figure 17: The plane modeled in MATLAB.**

If the system of equations (31) is constructed from these points and then solved by (38), then this system of equations have only one solution = 0. Another set of points (perturbation, Section 2.2.5 Perturbation in data) must be added to basic set of points. Points defined above and under main point have different values. We use the value –1 for points under the main set of points and 1 for points above. All added points had the same distance from main points. New points are standard defined in a direction of the normal in a vertex. The position in a direction of the normal has not been computed in this example yet. Points were localized in the direction of $z$-axis. It has been possible to evaluate the system of equations with new points and determine vector $x$. Vector $x$ contains values of weight variable $\lambda$.

```
FmodelDouble FmodelDouble::example(double x[])
{
 //x,y,z,value,id
 float points[][5] = {{ 0.0, 0.0, 0.5, 0.0, 1},{ 0.0, 2.0, 0.0, 0.0, 2},{ 0.0, 5.0, 0.0, 0.0, 3},
                      { 4.0, 5.0, 0.0, 0.0, 4},{ 3.0, 3.0, 1.5, 0.0, 5},{ 4.0, 1.0, 0.0, 0.0, 6},
                      { 4.0, 0.0, 0.0, 0.0, 7},{ 7.0, 0.0, 0.0, 0.0, 8},{ 7.0, 3.0,-0.5, 0.0, 9},
                      { 7.0, 5.0, 0.0, 0.0,10},{ 0.0, 0.0,-0.5,-1.0,11},{ 0.0, 2.0,-1.0,-1.0,12},
                      { 0.0, 5.0,-1.0,-1.0,13},{ 4.0, 5.0,-1.0,-1.0,14},{ 3.0, 3.0, 0.5,-1.0,15},
                      { 4.0, 1.0,-1.0,-1.0,16},{ 4.0, 0.0,-1.0,-1.0,17},{ 7.0, 0.0,-1.0,-1.0,18},
                      { 7.0, 3.0,-1.5,-1.0,19},{ 7.0, 5.0,-1.0,-1.0,20},{ 0.0, 0.0, 1.5, 1.0,21},
                      { 0.0, 2.0, 1.0, 1.0,22},{ 0.0, 5.0, 1.0, 1.0,23},{ 4.0, 5.0, 1.0, 1.0,24},
                      { 3.0, 3.0, 2.5, 1.0,25},{ 4.0, 1.0, 1.0, 1.0,26},{ 4.0, 0.0, 1.0, 1.0,27},
                      { 7.0, 0.0, 1.0, 1.0,28},{ 7.0, 3.0, 0.5, 1.0,29},{ 7.0, 5.0, 1.0, 1.0,30}
                     };

 float lambda[] = {-2.3295172376799261e-002,-1.8481464722608840e-002, 2.5858377867091463e-003,
                    2.1098099868712271e-004, 4.5557184756117988e-002, 1.9505919630826790e-002,
                   -4.8394018313553805e-002,-5.3201720212858912e-002,-4.9228452041243036e-002,
                   -2.5564906118562439e-002, 2.2050751724291642e-003, 2.1686611006784817e-002,
                   -4.1000618844257014e-003, 3.2915120419754820e-003,-1.4941496181109140e-001,
                   -1.8345689212648533e-002, 2.9941739703427690e-002, 2.9289512608919158e-002,
                    3.4054873609851606e-002, 6.2412410748465081e-003,-6.2054661325396700e-003,
                    6.3385063208558537e-002,-1.2180805206499088e-002, 7.4572906296871591e-002,
                   -6.6696489424080754e-002, 1.0218582605846335e-001,-1.0309468316983901e-002,
                    1.2792126679700517e-002, 4.5181944283673593e-002,-1.5108476079092338e-002
                  };

  float xx,yy,zz,r,value;
  value = 0.0;
  for(int i=0;i<30;i++)
  {
    xx = (x[0] - points[i][0])*(x[0] - points[i][0]);
    yy = (x[1] - points[i][1])*(x[1] - points[i][1]);
    zz = (x[2] - points[i][2])*(x[2] - points[i][2]);
    r = sqrt(xx+yy+zz);
    value += lambda[i]*((r*r)*log(r));
  }
 return(value);
}
```

**Figure 18: Example of the object in the CHF system.**

For the visualization of this example, module in MVE was used and this example was written as the function in our CHF system [Figure 18].

It can be seen, that the function $r^2 \log r$ was used as the basic function $\phi$ and the value of the object was calculated by equation (29). The visualization module in MVE requested this function at the value in spatial coordinates.

Figure 19 shows the final visualization of this example with the marching triangle method.

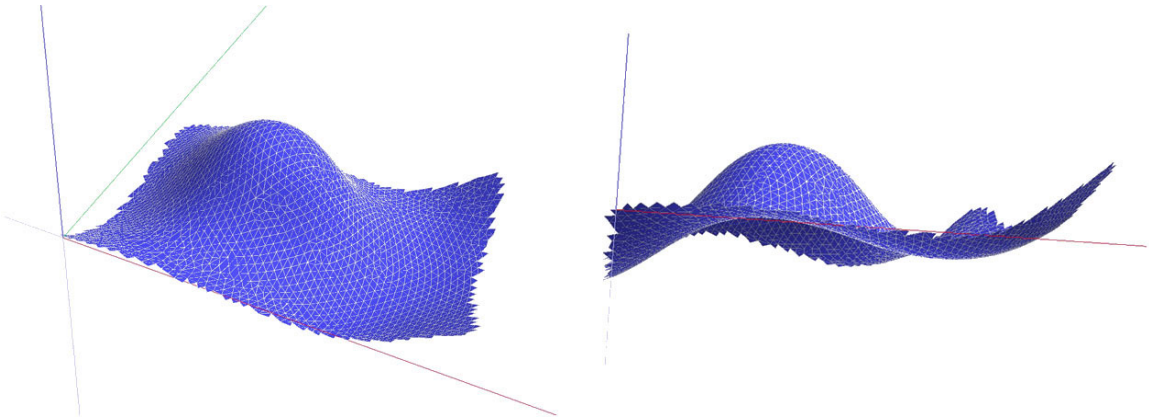


**Figure 19: The plane visualized with MVE module and the marching triangle method.**

### Example B

In this example the RBF method at the 2-d line was tested. A polygonal line was defined and how the RBF method makes approximation of this line was tested. Different basis functions and different methods for the off-surface points definition were tested [Figure 20]. Note that we will use the term 'off-surface point' everywhere despite of that in this example a polygonal line is the basic object.

Left image at Figure 20 shows off-surface points defined in a direction of *y*-axis and the right image shows definition of off-surface points in the direction of the vertex normal.

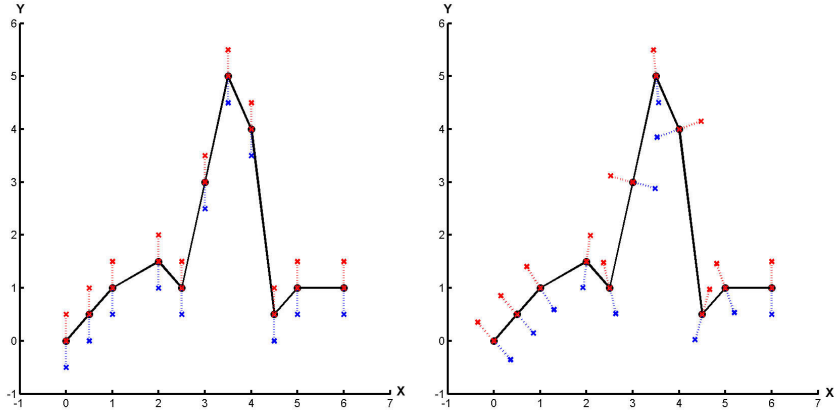



**Figure 20: Difference in off-surface points definition.**

The definition of off-surface points affects the form of the final approximation. The equation (32) is evaluated in each vertex grid on selected interval for the visualization of approximation with RBF method. The value in each vertex of this grid defines the „height" in *z*-axis. It can be seen from Figure 21 that the approximation of the polynomial line is better, if off-surface points are defined in the direction of the vertex normal. In this case $\phi = r^2 \log r$ is the basic function.

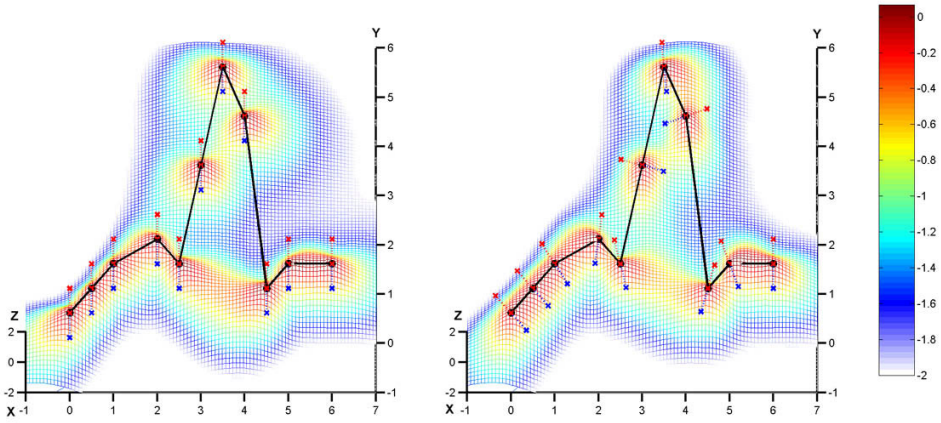



**Figure 21: The surface is a visualization of values in vertices of grid.**

It is necessary to note, that the polynomial line presented above was selected because you can see behavior of the RBF method if there are fast changes on the line. Now we can take a look at line approximation if the line has "nice" course.

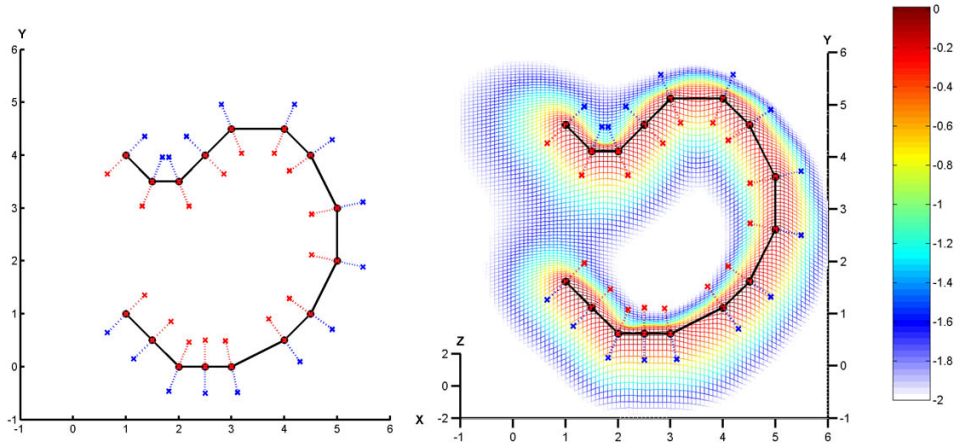



**Figure 22: Approximation of the line without any quick changes.**

The line has really nice course. The final curve is the minimal value on the interval from off-surface points at the right side (blue) and point at the left side (red). There can be used different algorithms for the visualization. There is used stepping algorithm, which provides dividing of the interval between red and blue points, for each segment of the polygonal line. There are calculated values of equation (32) on this interval and in the direction of the segment normal the minimal value is searched for. These values represent points on the final curve. Left image at Figure 23 shows the visualization of the final curve in 2-d. The right image, at the same figure, shows values of the equation (32) in direction of *z*-axis. Note, that for the visualization, the basic stepping algorithm from the marching triangles method can be also used.

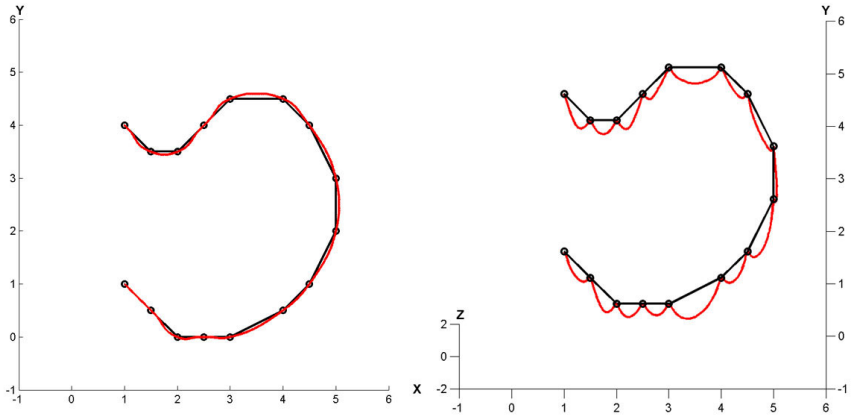



**Figure 23: Final curve (red) and original polygonal line (black).**

The basic function has influence on the course of the curve in 2-d and the course of the surface in 3-d. The group of basic functions is different for 2-d and 3-d approximation. Rather, the functions usable in 2-d case can be used in 3-d whereas in the inverse case it does not apply. From the group of functions introduced in section 2.2.6 only functions b) and c) can be used. The functions were used at the polynomial line shown on Figure 22. Figure 24 demonstrates the influence of basic function of final result. All functions presented in section 2.2.6 have been used.

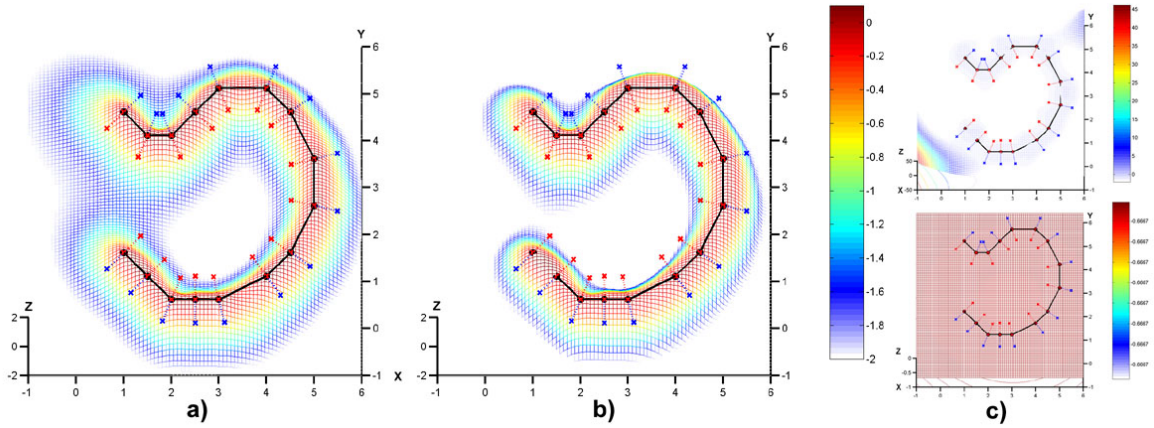



**Figure 24: Different basic function using. a) Thin-plate (2-d), b) Thin-plate (3-d), c) Compactly Supported and Gaussian.**

The basic function selection change the course at the interval defined by the off-surface points. From images c) compactly supported functions and Gaussian functions for approximation in 2-d evidently cannot be used.

The purpose of all these tests was to demonstrate possible modification of the RBF. The last figure (Figure 25) shows the part of the MATLAB source. Parts with the construction of the matrix system and the evaluation of the matrix system with LU factorization are shown.

```
% matrix system construction
for i=1:1:a_row,
 for j=1:1:a_row,
  xi = [BODY_ALL(i,1),BODY_ALL(i,2)];
  xj = [BODY_ALL(j,1),BODY_ALL(j,2)];
  r = sqrt((xi(1)-xj(1))^2+(xi(2)-xj(2))^2);
  if(r == 0)
   val = 0;
  else
    val = abs(r)^2*log(abs(r));
  end;
  A(i,j) = val;
 end;
end;

% if polynomial p(x) is used
if(polynomial == 1)
 a_row = a_row + 3;
 for i=1:1:a_row-3,
  A(i,a_row-2) = BODY_ALL(i,1);
  A(i,a_row-1) = BODY_ALL(i,2);
  A(i,a_row)   = 1;
 end;

 for j=1:1:a_row-3,
  A(a_row-2,j) = BODY_ALL(j,1);
  A(a_row-1,j) = BODY_ALL(j,2);
  A(a_row,j)   = 1;
 end;
end;

% LU factorization
[L,U] = lu(A);
lambda = U\ (L\b);
```

**Figure 25: Creating and solving of the matrix system in MATLAB.**

## Conclusion and Future Work

Implicitly defined object modeling starts to be very popular part of computer graphics. This modeling has many faces. In this work a complex method for implicitization of the object defined by the triangular mesh or point-cloud data was introduced. The RBF method has very many aspects for improvements and future exploration. Final implicit description and final quality of visualized object depends on the parameters of the RBF method. The basic parameters are: selection of the basic function and its influence to accuracy of the approximation, which method for solving of the linear equation system is better with respect to the algorithm, time complexity and the off-surface points definition and their influence to the approximation. All these parts are important and have very big influence to the final objects and the overall complexity of the method. These parameters will be scrutinized and described in further work.

It has been tested and implemented the RBF method for a case of 2-d polynomial line in our previous work. Now we are going to continue with 3-d objects, where the number of points together with the algorithm complexity can make such method unusable. There were presented methods for speedup of the RBF method in the text, but these methods can affect accuracy of the final object. Parallelization can be also used. The RBF method can be parallelized in several steps. Parallel method can be used for solving the linear equation system and for evaluation of the final surface if the compactly supported radial basis functions are used. The accuracy of the final object and parallelization will be subject of further research too.

The part of our previous work covered Constructive Solid Geometry (CSG) and the modeling system which provides direct compilation of an implicitly defined object. The principle of direct compilation was introduced in an example A. We want to extend the possibilities of our system for direct compilation of complex models in further research. We also want to continue with the modeling system and compilation of the models obtained from the RBF method. Within the CSG framework the RBF offers a new way of modeling real-world (scanned) objects since it is inherently a solid model. It can be manipulated through a series of Boolean unions and intersections with other objects in a manner similar to how simpler geometric primitives are currently used to construct more complicated objects.

Summary, the aims of doctoral thesis are:

- to explore influence of the basic functions to the accuracy of the approximation and propose functions suitable for different objects
- to explore off-surface points definition and their influence to the approximation and propose off-surface points selection for noisy and quickly changing data
- to explore methods for solving the linear equation system in relation to parallelization
- to implement stable and robust RBF method with possibility to compile objects or the CSG tree structure

## References


[Adzhiev99] Adzhiev, V., Cartwright, R., Fausett, E., Ossipov, A., Pasko, A., Savchenko V.: "HyperFun Project: a Framework for Collaborative Multidimensional F-rep Modeling", Proceedings of a Eurographics & ACM SIGGRAPH Workshop "Implicit Surfaces '99", ed. J.Hughes and C.Schlick, Bordeaux, France, pp.59-69, 1999.

[Adzhiev00] Adzhiev, V., Kazakov, M., Pasko, A., Savchenko, V.: "Hybrid System Architecture for Volume Modeling", Computers and Graphics, vol. 24, No. 1, pp. 67-78, 2000.

[Angelidis02] Angelidis, A., Jepp, P., Cani, M.P.: "Implicit Modeling with Skeleton Curves: Controlled Blending in Contact Situations", Shape Modeling International, 2002.

[ARANZ] Applied Research Associates NZ Ltd: http://aranz.com/

[Bastl01] Bastl, B.: Metody moderni algebry a jeji aplikace, Thesis, University of West Bohemia, Faculty of Applied Sciences, Department of Mathematics, Czech Republic, 2001. (Czech language)

[Bastl03] Bastl, B.: Metody eliminace promennych pro soustavy nelinearnich algebraickych rovnic a jejich aplikace v geometrickem modelovani, State of the Art and Concept of Doctoral Thesis, University of West Bohemia, Faculty of Applied Sciences, Department of Mathematics, Czech Republic, 2003. (Czech language)

[Beatson99] Beatson, R. K., Cherrie, J. B., Mouat, C. T.: "Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration", Advances in Computational Mathematics, vol. 11, pp. 253-270, 1999.

[Beatson00] Beatson, R. K., Light, W.A., Billings, S.: "Fast solution of the radial basis function interpolation equations: Domain decomposition methods", SIAM J. Sci. Comput., vol. 22, no. 5, pp. 1717-1740, 2000.

[Berchtold00] Berchtold, J.: "The Bernstein Basis in Set-Theoretic Geometric Modelling", Thesis, Department of Mechanical Engineering, University of Bath, 2000.

[Bloomenthal88] Bloomenthal, J.: "Polygonizaion of Implicit Surface", Computer-Aided Geometric Design, vol. 5, no. 4, pp. 341-355, 1988.

[Bloomenthal91] Bloomenthal, J., Shoemake, K.: "Convolution surfaces", Computer Graphics, 25(4):251-256, 1991.

[Bloomenthal97] Bloomenthal, J., Bajaj, C., Blinn, J., Cani-Gascuel, M. P., Rockwood, A., Wyvill, B., Wyvill, G.: "Introduction to Implicit Surfaces", Morgan Kaufmann, 1997.

[Carr97] Carr, J. C., Fright, W. R., Beatson, R. K.: "Surface interpolation with radial basis functions for medical imaging", IEEE Trans. Medical Imaging, vol. 16, pp. 96-107, February 1997.

[Carr01] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., Evans, T. R.: "Reconstruction and representation of 3D objects with radial basis functions", Computer Graphics (SIGGRAPH 2001 proceedings), pp. 67-76, August 2001.

[Cermak02a] Čermák M., Skala V. (2002). Polygonization by the Edge Spinning. Algoritmy 2002 Conf.proceedings, Univ.of Technology, Slovakia, ISBN 80-227-1750-9, pp.245-252, 2002.

[Cermak02b] Čermák M., Skala V. (2002). Accelerated Edge Spinning Algorithm for Implicit Surfaces. ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.174-179, 2002.

[Dekkers97] Dekkers, K., Overveld, van, Golsteijn, R.: "Combining CSG Modeling with Soft Blending using Lipschitz-based Implicit Surfaces", Technical Report, Eindhoven University of Technology, Computer Graphics Group, 1997.

[Franc02] Franc, M.: Methods for Polygonal Mesh Simplification. State of the Art and Concept of Doctoral Thesis, Technical Report No. DCSE/TR-2002-01, University of West Bohemia, Plzen, Czech Republic, January 2002.
[Gallo90] Gallo, G., Mishra, B.: “Efficient algorithms and bounds for Wu-Ritt characteristic sets”, In Proc. MEGA'90, pages 119--142, 1990.
[Gallo91] Gallo, G., Mishra, B.: “Wu-Ritt characteristic sets and their complexity. In Computational Geometry: Papers from the DIMACS Special Year, volume 6, pages 111-136. AMS and ACM, 1991.
[Greengard87] Greengard, L., Rokhlin, V.: “A fast algorithm for particle simulations”, J. Comput. Phys., 73(2):325--348, 1987.
[Hart96] Hart, J.C.: “Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces”, The Visual Computer 12 (10), Dec. 1996, pp. 527-545.
[Hartmann98] Hartmann, E.: “A marching method for the triangulation of surfaces”, The Visual Computer 14, 3, 95-108, 1998.
[Hoffmann93] Hoffmann, C.M.: “Implicit Curves and Surfaces in CAGD”, IEEE Computer Graphics and Appl. 13, Jan. 1993, 79-88.
[Hoppe92] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: “Surface reconstruction from unorganized points”, Computer Graphics (SIGGRAPH '92 Proceedings), 26(2):71--78, July 1992.
[Hubb98] Huub van de Wetering, Martijn de Kort, Kees van Overveld: "Scan-conversion of implicit surfaces with Lipschitz condition", in: proceedings of The Third International Workshop on Implicit Surfaces, June 15-16, Seattle, USA, 1998.
[HyperFun99] HyperFun: Language for F-rep Geometric Modeling, http://www.hyperfun.org, 1999.
[Mika96] Míka, S.: “Numerické metody Lineární algebra”, ZČU Plzeň, 1996. (Czech language)
[Morse01] Morse, B., Yoo, T. S., Rheingans, P., Chen, D. T., Subramanian, K.R.: “Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions”, in Proceedings of the Shape Modeling conference, Genova, Italy, 89-98, May 2001.
[Overveld93] C.W.A.M. van Overveld, Wyvill, B.: ”Shrinkwrap: an adaptive algorithm for polygonizing an implicit surface”, Research Report No. 93/514/19, March 1993, The University of Calgary, Calgary, Alberta, Canada; presented at the Western Canadian Conference on Computer Graphics ‘SkiGraph’, Silver Star, BC, Ca., 1993.
[Ohtake01] Ohtake, Y., Belyaev, A. G., Pasko, A.: ”Dynamic meshes for accurate polygonization of implicit surfaces with sharp features”, In Shape Modeling International 2001, pages 74--81, Genova, Italy, May 2001.
[Pasko93] Pasko, A., Savchenko, V., Adzhiev, V., Sourin, A.: “Multidimensional geometric modeling and visualization based on the function representation of objects”, Technical Report 93-1-008, University of Aizu, Japan, 1993.
[Pasko95] Pasko, A., Adzhiev, V., Sourin, A., Savchenko, V.: “Function Representation in Geometric Modeling: concepts, Implementation and Applications”, The Visual Computer, 8 (2), pp. 429--446, 1995.
[Rege96] Rege, A.G.: “A Toolkit for Algebra and Geometry”, Ph.D thesis, Computer Science of University of California at Berkeley, 1996.
[Rektorys95] Rektorys, K.: “Přehled užité matematiky I,II“, Nakladatelství Prométheus, Praha, 1995. (Czech language)
[Rigaudiere99] Rigaudiere, D., Gesquiere, G., Faudot, D.: “New Implicit Primitives Used in Reconstruction by Skeletons”, WSCG’99, 1999.

[Shersyuk99] Sherstyuk, A.: “Kernel functions in convolution surfaces: a comparative analysis”, The Visual Computer, 15(4), 1999.
[Turk99] Turk, G., O'Brien, J.: “Shape Transformation Using Variational Implicit Functions”, SIGGRAPH 99, August 1999, pp. 335-342, 1999.
[Turk01] Turk, G., Dinh, H. Q., O'Brien, J., Yngve, G.: “Implicit Surfaces that Interpolate”, Shape Modelling International 2001 Genova, Italy, May 7-11, pp. 62-71, 2001.
[Turk02] Turk, G., O'Brien, J.F.: “Modelling with Implicit Surfaces that Interpolate”, ACM Transactions on Graphics, Vol. 21, No. 4, pp. 855-873, October 2002.
[Uhlir01] Uhlíř, K., Skala, V.: “Interaktivní system pro generovaní implicitních funkcí a jejich modelování”, Thesis, University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering, Czech Republic, 2001. (Czech language)
[Uhlir02] Uhlir, K., Skala, V.: “Kompilovaný HyperFun”, Technical Report DCSE/TR-2002-07, University of West Bohemia, Czech Republic, 2002. (Czech language)
[Uhlir03] Uhlir, K., Skala,V.: “The Implicit Function Modelling System - Comparison of C++ and C# Solutions”, C# and .NET Technologies'2003, University of West Bohemia, Czech Republic, ISBN 80-903100-3-6, 2003.
[Velho96] L. Velho.: “Simple and efficient polygonization of implicit surfaces”, Journal of Graphics Tools, 1(2):5-24, 1996.
[Wendland95] Wendland, H.: “Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree”, Advances in Computational Mathematics 4 (1995), 389-396, 1995.
[Wyvill86] Wyvill, B., McPheeters, C., Wyvill, G.: “Data structure for soft objects”, The Visual Computer, 2(4):227--234, 1986.
[Wyvill87] Wyvill, G., Ward, A., Brown, T.: “Sketching by Ray Tracing”, in Computer Graphics 1987, New York, Springer-Verlang, 1987, pp. 315-333 (Proceedings of Computer Graphics International).
[Yngve02] Yngve, G., Turk, G.: “Robust Creation of Implicit Surfaces from Polygonal Meshes”, IEEE Transactions on Vizualization and Computer Graphics, Vol. 8, No. 4, pp. 346-359, October-December 2002.
[Yoo01] Yoo, T. S., Morse, B., Subramanian, K., Rheingans, P., Ackerman, M. J.: “Anatomic modeling from unstructured samples using variational implicit surfaces”, Medicine Meets Virtual Reality, 2001.

# Appendix A

## Publications


[i] Uhlir, K., Skala,V.: The Implicit Function Modelling System - Comparison of C++ and C# Solutions , C# and .NET Technologies'2003, University of West Bohemia, Czech Republic, ISBN 80-903100-3-6, 2003.

[ii] Uhlíř, K., Skala, V.: Kompilovaný HyperFun , Technical Report DCSE/TR-2002-07, University of West Bohemia, Czech Republic, 2002. (Czech language)

[iii] Uhlíř, K., Skala, V. (supervisor): Interaktivní system pro generovaní implicitních funkcí a jejich modelování , Thesis, University of West Bohemia, Czech Republic, 2001. (Czech language)


## Stays and Conferences

**Stays:**

12.2.2000 – 15.9.2000 University of Ioannina, Greece, Erasmus/Socrates project

**Conferences:**

5.2.2003 – 8.2.2003 C# and .NET Technologies'2003, Czech Republic, [i]